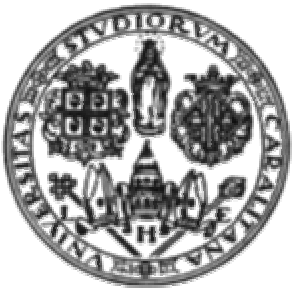

UNIVERSITÀ DEGLI STUDI DI CAGLIARI
FACOLTÀ DI SCIENZE MM. FF. NN.
CORSO DI LAUREA IN FISICA

PROGETTO DI UNA MACCHINA MONTE CARLO DEDICATA ALLA SOLUZIONE DI PROBLEMI DI FISICA PER LA RADIOTERAPIA

Relatore :

Prof. Paolo Randaccio

Dott. Massimiliano Porcu

Tesi di Laurea di :

Vincenzo De Leo

ANNO ACCADEMICO 2000 / 2001

Ai miei genitori,

per avermi insegnato il valore dello studio…

Ringraziamenti

È con grande piacere che ringrazio Billy (alias Massimiliano Porcu) per la pazienza e la simpatia che mi ha dimostrato in questi mesi in cui abbiamo lavorato assieme, mio fratello Roberto per avermi trasmesso la sua passione per lo studio della fisica e Laura per avermi sostenuto sempre nei momenti più difficili col suo affetto.

Non possono mancare, inoltre, i ringraziamenti a tutto il gruppo del laboratorio di fisica medica (in particolare i miei compagni di sventura Daniela e Franco), al gruppo della Comunità Missionaria di Villaregia e a tutti coloro che mi hanno accompagnato durante questi anni di studio universitario.

Infine, ma non per ultimo, un grazie particolare va al Professor Paolo Randaccio per avermi seguito sempre con dedizione e interesse, mostrandosi sempre pronto a darmi utili consigli e incoraggiamenti.

Indice generale

Introduzione

Capitolo 1

Problema del calcolo della dose in radioterapia

Capitolo 2

Il metodo Monte Carlo e la sua applicazione allo studio di sistemi fisici reali.

Capitolo 3

Un software per il calcolo del tracciamento

Capitolo 4

Field Programmable Gate Array (FPGA)

Capitolo 5

Progetto del simulatore Monte Carlo Hardware

Capitolo 6

Confronto tra software e hardware

Conclusioni

Appendice A

Appendice B

Bibliografia

Introduzione

Le radiazioni ionizzanti, costituiscono uno strumento importante e ampiamente utilizzato nella terapia dei tumori. Attraverso l'irradiazione della regione affetta da tumore,infatti, è possibile debellare le cellule neoplastiche o limitarne lo sviluppo; purtroppo, quest'azione non è selettiva e crea danni anche al tessuto sano circostante la regione tumorale. L'obiettivo primario del radioterapista è dunque quello di rendere massimo il "beneficio" del trattamento, ossia il rapporto tra il numero di cellule malate distrutte e quelle non danneggiate dall'irraggiamento. Nella gran maggioranza dei casi, i trattamenti radioterapici sono effettuati con l'ausilio di due tipi di radiazioni ionizzanti: i fotoni e gli elettroni; queste sono prodotte principalmente con l'utilizzo di acceleratori lineari e, raramente oramai, tramite radioisotopi (^{60}C, ^{137}Cs e ^{192}Ir), con energie che vanno da poche centinaia di KeV a 50 MeV. Per realizzare il trattamento desiderato, il medico può agire su due parametri del fascio prodotto: la forma geometrica, che determina l'estensione della regione irradiata, e l'energia.

La grandezza fisica fondamentale in questo campo è la *dose assorbita*, definita come l'energia depositata dalla radiazione per unità di massa del mezzo irradiato: i dan-

danni prodotti al tessuto biologico sono funzione di questa grandezza, oltre che del tipo specifico di tessuto. Viene detto *piano di trattamento*, il processo tramite il quale si realizza una stima preventiva della dose assorbita, punto per punto, della zona irradiata.

La compilazione del piano di trattamento costituisce un passo cruciale del lavoro del radioterapista: attraverso esso, infatti, è possibile prevedere l'entità degli effetti che si produrranno e calibrare l'azione in maniera da avere il massimo vantaggio possibile. Per essere considerato accettabile, un piano di trattamento deve soddisfare due requisiti cruciali: la precisione nella stima della dose assorbita e la velocità di compilazione; dal primo dipende la possibilità di limitare il più possibile gli effetti collaterali del trattamento o, alternativamente, di massimizzarne l'incisività e il secondo consente all'operatore di realizzare cambiamenti nell'impostazione della terapia e di avere, in "tempo reale", il prospetto degli effetti che questi produrranno.

In questo contesto si inserisce il mio lavoro, che nasce dalla necessità di fornire al medico Radioterapista uno strumento che gli consenta la realizzazione di piani di trattamento, con una precisione che sia quella sufficiente alla corretta elaborazione dei parametri del fascio di radiazioni nel particolare caso affrontato, e che applichi il concetto di "tempo reale"; oggi, infatti, i software utilizzati per queste applicazioni sfruttano delle tecniche empiriche che, sebbene offrano una notevole efficienza, sono carenti per la qualità dei dati elaborati: questa situazione non può essere certo ottimale per il medico che vuole incidere con decisione sul tumore minimizzando i rischi di danneggiare le cellule vitali che si trovano nelle vicinanze della zona irradiata. Esiste, però, uno strumento alternativo alle tecniche empiriche che è estremamente preciso ma molto poco efficiente: il metodo Monte Carlo.

I progressi dell'elettronica e la possibilità di poter disporre di calcolatori sempre più potenti con spese minime hanno aperto la strada alla simulazione numerica: tramite il computer è, infatti, possibile creare un modello di un sistema fisico reale, descriverlo attraverso delle equazioni e simularne l'evoluzione, realizzando una sorta di *esperimento virtuale*. Tra le metodologie di simulazione numerica si colloca anche il metodo Monte Carlo, il quale fa ricorso a strumenti statistici per determinare l'evoluzione del sistema e l'elaborazione delle informazioni. La poca efficienza di questo metodo è dovuta al fatto che l'evoluzione del sistema macroscopico è deter-

minata dal comportamento dei singoli oggetti che lo compongono a livello microscopico: se il numero di questi oggetti è molto grande, dunque, è molto grande anche il numero di calcoli che devono essere eseguiti ogni volta che si vuole conoscere la nuova configurazione del sistema dopo un certo evento; il tempo necessario alla conclusione dell'intero processo può arrivare anche a 500 ore.

In un Monte Carlo per il calcolo della dose assorbita è necessario realizzare il *trasporto della radiazione* nel mezzo biologico, calcolando l'energia depositata lungo il percorso. Per avere una stima precisa dell'energia depositata occorre, di norma, simulare la propagazione di un numero di particelle compreso tra 10^6 e 10^8 interagenti in volumetti di tessuto di pochi mm^3. Ciò comporta un tempo di calcolo dell'ordine delle ore/decine di ore. Il Monte Carlo si presenta, quindi, come un metodo molto preciso ma praticamente inapplicabile per la compilazione dei piani di trattamento.

Parallelamente allo sviluppo dei calcolatori ad uso generico, però, la ricerca elettronica si sta sviluppando anche nel campo delle applicazioni scientifiche: è oggi possibile acquistare, infatti, a costi accessibili, delle logiche programmabili direttamente dall'utente, dette FPGA (Field Programmable Gate Array), con le quali si possono realizzare circuiti logici complessi in maniera facile e veloce.

L'idea che abbiamo allora voluto sviluppare in questo lavoro è stata quella di sfruttare l'elasticità di programmazione offerta dalle FPGA per realizzare il progetto di un circuito logico integrato dedicato alla soluzione di questi problemi: l'obiettivo a lungo termine è quello di utilizzare le potenzialità di questi nuovi circuiti microelettronica programmabili per abbattere i tempi di esecuzione e rendere il metodo Monte Carlo concretamente utilizzabile per la compilazione dei piani di trattamento.

Il lavoro da me svolto è da considerarsi uno studio preliminare sulle possibilità offerte dal filone di ricerca; esso è un primo approccio alla metodologia con cui si può riformulare in termini "hardware" problemi che tradizionalmente sono stati affrontati e risolti via "software".

Nei capitoli che seguono è affrontato il problema del calcolo della dose assorbita in radioterapia (Capitolo 1), sono definite le caratteristiche fondamentali del metodo Monte Carlo (Capitolo 2), è descritto un software da me realizzato in C per l'applicazione del metodo Monte Carlo nel calcolo della dose (Capitolo 3), viene

mostrata l'architettura di una FPGA e, in particolare, quella della Spartan II che è stata adottata per la realizzazione del progetto (Capitolo 4), è descritta la realizzazione in hardware di un algoritmo per calcolo della dose analogo a quello sviluppato in C (Capitolo 5) e sono riportate le misure sperimentali del tempo di esecuzione del software e della sua rielaborazione hardware (Capitolo 6).

Grazie ai dati ricavati abbiamo potuto effettuare una valutazione significativa del guadagno che si è ottenuto, in termini di tempo di calcolo, grazie alla macchina realizzata nel mio lavoro; di questo parleremo nelle conclusioni finali.

Capitolo 1

Problema del calcolo della dose in radioterapia

1.1 Introduzione.

Lo scopo della radioterapia è quello di bloccare la crescita di un tumore o impedirne lo sviluppo ed evitare contemporaneamente di produrre danni irreversibili ai tessuti sani; le tecniche radioterapiche si dividono in due classi: la impianto-terapia e la tele-terapia:

- nelle terapie di impianto la sorgente radiogena è posta a diretto contatto con il tumore; è possibile distinguere tra impianto-terapia superficiale, endocavitaria e interstiziale; vengono utilizzate sorgenti isotopiche di attività relativamente bassa (1 mCi ÷ 100 mCi) con tempi di trattamento lunghi, anche di alcuni giorni;
- nella tele-terapia la sorgente viene posta a una distanza notevole dal corpo del paziente; sono usate sorgenti isotopiche di attività elevata (1000 Ci ÷ 10000 Ci) oppure apparecchi con intensità di emissione elevata come gli acceleratori lineari o i tubi a raggi X; i tempi di trattamento sono brevi, dell'ordine di qualche minuto.

Le radiazioni ionizzanti sono state impiegate sin dall'epoca di Roentgen e Curie per la cura dei tumori. È stato notato che le radiazioni riescono a ridurre la crescita dei tessuti neoplastici (tumori) senza comportare danni permanenti ai tessuti sani: ciò è dovuto al fatto che i tessuti più sensibili alle radiazioni sono quelli a rapida crescita, nei quali le cellule si trovano frequentemente in fase mitotica. Un danno al DNA di una cellula in mitosi si traduce in morte della cellula stessa, ovvero nell'impossibilità della duplicazione; anche i tessuti sani vengono danneggiati, ma quasi sempre il danno è temporaneo.

Poiché il danno biologico è legato alla quantità di radiazioni ionizzanti assorbita dal paziente, è di fondamentale importanza definire delle grandezze fisiche che diano una misura di tale quantità; le più significative per le misure di dose in ambito radiologico sono la dose assorbita, la dose equivalente e la dose efficace.

La grandezza dosimetrica fondamentale in radioprotezione è la dose assorbita D, definita come l'energia assorbita per unità di massa:

$$D = \frac{dE}{dm} \qquad (1.1)$$

Il dE che compare nella definizione di D rappresenta l'energia media impartita dalla radiazione ionizzante alla materia mentre dm è la massa del volume di materia irradiato. L'unità di misura nel S.I. è il Gray (Gy), che indica la quantità di energia assorbita per unità di massa; è definito come:

$$1\mathrm{Gy} = \frac{1\mathrm{J}}{1\mathrm{Kg}} \qquad (1.2)$$

Una unità di misura vecchia, non più usata, è il rad:

$$1\mathrm{rad} = \frac{1}{100}\mathrm{Gy} \qquad (1.3)$$

Un'altra unità di misura, il Roentgen, si riferisce invece alle misure di esposizione da radiazione, e non di assorbimento. Esso indica il tasso di esposizione, equivalente alla quantità di ionizzazione prodotta in aria da raggi X e raggi γ.

In certe applicazioni però, piuttosto che la dose in un punto, riveste maggior interesse la dose assorbita in media da un tessuto o da un organo, corretta per la qualità della radiazione, chiamata dose equivalente (H_t):

$$H_t = \sum_R w_R \cdot D_{t,R} \qquad \textbf{(1.4)}$$

dove $D_{t,R}$ è la dose di radiazione del tipo R mediata sul tessuto t e w_R è il fattore di ponderazione per la radiazione; il danno biologico causato da una radiazione dipende infatti dall'energia della radiazione e dal tipo di radiazione: ad esempio, assorbire una certa dose di particelle α produce un danno maggiore che assorbire una uguale dose di protoni, e questa un danno ancora maggiore di una uguale dose di elettroni o raggi γ. Il valore del fattore di ponderazione per la radiazione (w_R) che si applica ad una radiazione di un determinato tipo ed energia è stato scelto dalla commissione ICRP (International Commission on Radiological Protection), una commissione fondata nel 1928 a seguito di una decisione del 2° Congresso Internazionale di Radiologia e che si avvale dei progressi compiuti dalle più importanti organizzazioni nazionali in materia di radioprotezione, con l'intento che esso sia rappresentativo dei valori di efficacia biologica relativa di quella radiazione in grado di indurre effetti stocastici a basse dosi; l'efficacia biologica relativa (RBE) di una radiazione rispetto ad un'altra è il rapporto inverso tra le dosi assorbite che producono il medesimo grado di un ben definito effetto biologico.

I valori di w_R sono grossolanamente compatibili con i valori della qualità di radiazione (Q), un parametro che tiene conto della pericolosità delle varie radiazioni rispetto a una radiazione di riferimento (fotoni) a cui viene assegnato per definizione un Q uguale a 1; a loro volta, i valori di Q sono correlati con il trasferimento lineare di energia (LET), una grandezza che misura la densità di ionizzazione lungo la traccia di una particella ionizzante. Originariamente questa relazione doveva soltanto rappresentare un'indicazione grossolana della variazione dei valori di Q per le diverse radiazioni; si è scelto in seguito di attribuire il valore 1 a tutte le radiazioni a basso LET, inclusi i raggi X e gamma di ogni energia. I valori scelti per le altre radiazioni si fondano su valori sperimentalmente osservati di RBE, indipendentemente dal fatto che la radiazione di riferimento sia X o gamma.

Quando il campo di radiazioni è composto da tipi ed energie diversi, ciascuno con diversi valori di w_R, la radiazione assorbita deve essere suddivisa in blocchi, ciascuno con il suo proprio valore di w_R, sommati poi per dare la dose equivalente totale.

Riportiamo nella Tabella 1.1 un elenco dei fattori di ponderazione w_R per i vari tipi di radiazioni e per diversi intervalli di energia:

Tipo di radiazione ed intervallo di energia	**w_R**
Fotoni (tutte le energie)	1
Elettroni e **muoni** (tutte le energie)	1
Neutroni, energia <10 keV	5
10 keV - 100 keV	10
>100 keV - 2 MeV	20
>2 MeV - 20 MeV	10
>20 MeV	5
Protoni, tranne quelli di rinculo, energia > 2 MeV	5
Particelle **alfa**, **frammenti di fissione**, **nuclei pesanti**	20

Tabella 1.1

L'unità di misura della dose equivalente è il Sievert (Sv), definito come il Gray:

$$1\mathrm{Sv} = \frac{1\mathrm{J}}{1\mathrm{Kg}} \qquad (1.5)$$

Una unità di misura non più usata è il rem, definito come:

$$1\mathrm{rem} = \frac{1}{100}\mathrm{Sv} \qquad (1.6)$$

È importante specificare che la dose equivalente non è una misura diretta della dose assorbita.

La relazione tra la probabilità di effetti stocastici e dose equivalente dipende anche dall'organo o tessuto irradiato. Il fattore di peso utilizzato è chiamato fattore di ponderazione per i tessuti w_t e rappresenta il contributo relativo di quell'organo o tessuto al danno totale dovuto agli effetti stocastici per irradiazione uniforme su tutto il corpo (vedi Tabella 1.2). La dose equivalente pesata (dose assorbita pesata sui w_R e sui w_t) prende il nome di dose efficace (E):

$$E = \sum_t w_t \cdot H_t = \sum_t w_t \sum_R w_R \cdot D_{t,R} \quad (1.7)$$

L'unità di misura della dose efficace è il Sievert.

Tessuto o organo	**Fattore di ponderazione per i tessuti**
Gonadi	0,20
Midollo osseo	0,12
Colon	0,12
Polmone	0,12
Stomaco	0,12
Vescica	0,05
Mammella	0,05
Fegato	0,05
Esofago	0,05
Tiroide	0,05
Cute	0,01
Superfici ossee	0,01
Altri tessuti	0,05

Tabella 1.2

I valori sono stati derivati per una popolazione di riferimento, composta da un uguale numero di persone dei due sessi e con un ampio intervallo d'età. Nella definizione della dose efficace i valori si applicano ai lavoratori, alla popolazione e ad ambedue i sessi. Ai fini di calcolo, la categoria "Altri tessuti" comprende i seguenti tessuti ed organi: surrene, cervello, intestino crasso superiore, intestino tenue, rene, muscoli, pancreas, milza, timo e utero. Nel caso eccezionale in cui un singolo organo o tessuto incluso nella categoria "Altri tessuti", ricevesse una dose equivalente superiore alla dose massima ricevuta da uno dei dodici organi che hanno uno specifico valore W_T, si dovrà applicare un fattore di peso di 0,025 a quell'organo o tessuto ed un altro fattore di 0,025 alla dose media ricevuta dagli organi compresi nella categoria altri tessuti.

Per calcolare la distribuzione di dose assorbita all'interno del corpo del paziente sottoposto a radioterapia occorre conoscere le caratteristiche della sorgente, la forma e la posizione del tumore e dei vari tessuti circostanti interessati dal campo di radiazioni e la posizione reciproca tra la sorgente (o le sorgenti) e il corpo del paziente. Normalmente è preposto a questo compito un fisico che collabora con il medico radioterapista per predisporre il cosiddetto "piano di trattamento", cioè una stima preventiva della dose di radiazione assorbita punto per punto nella zona da irradiare; i calcoli sono effettuati quasi esclusivamente con l'impiego di elaboratori. La successione delle operazioni necessarie alla creazione di un corretto piano di trattamento sono le seguenti:

1) I dati del paziente sono ottenuti tramite tomografie ;
2) Il medico individua la posizione e il tipo di tumore e stabilisce la dose da somministrare;
3) Il fisico individua la posizione della sorgente (o sorgenti) e la durata dell'irraggiamento;
4) Attraverso un opportuno programma si elabora il piano di trattamento e si producono degli elaborati grafici con l'indicazione dei livelli di dose nei vari punti, tramite una rappresentazione a curve di isodose;
5) In genere si procede a una ottimizzazione per ridurre la dose ai tessuti sani radiosensibili e si ripetono le fasi 3) e 4).

1.2 Compilazione dei piani di trattamento

La compilazione del piano di trattamento costituisce un passo cruciale del lavoro del radioterapista: attraverso esso è infatti possibile prevedere l'entità degli effetti che si produrranno e calibrare l'azione in maniera da avere il massimo vantaggio possibile. Affinché un piano di trattamento sia accettabile deve soddisfare due requisiti indispensabili: deve essere sufficientemente preciso nella stima della dose di radiazione assorbita dal paziente e deve essere disponibile al radioterapista in tempi ragionevoli; la prima è necessaria affinché vengano limitati il più possibile gli effetti collaterali del trattamento, cioè il danneggiamento delle cellule sane nella regione prossima al tumore, e la seconda permette al radioterapista di effettuare modifiche nell'impostazione del piano di trattamento ottenendo in tempo reale il prospetto degli effetti che questi cambiamenti comporteranno: uno strumento che consenta la compilazione del piano di trattamento rispettando queste caratteristiche permette al radioterapista di effettuare molti tentativi, variando di volta in volta i parametri del fascio da inviare al paziente, al fine di migliorarne gli effetti sulla zona interessata, in un tempo sufficientemente breve da non causare l'espandersi del tumore verso le regioni circostanti e quindi un aggravamento della situazione del paziente.

Nella compilazione del piano di trattamento sono utilizzate tipicamente delle tecniche empiriche, le quali costituiscono un compromesso tra le esigenze contrastanti di efficienza e precisione; infatti, per quanto queste tecniche consentano di ottenere rapidamente dei risultati, questi sono spesso di bassa qualità; in queste condizioni il radioterapista non può agire incisivamente sul tumore come vorrebbe. Esiste però uno strumento alternativo alle tecniche empiriche che è estremamente preciso ma molto poco efficiente: il metodo Monte Carlo.

1.3 Il metodo Monte Carlo per la compilazione dei piani di trattamento.

Il metodo Monte Carlo è una soluzione numerica a un problema in cui gli oggetti in esame interagiscono con altri oggetti o il mezzo in cui si trovano tramite semplici relazioni oggetto-oggetto o oggetto-ambiente: si può dire che questo è un tentativo di rappresentare la natura attraverso la simulazione diretta della dinamica essenziale del sistema in questione. In questo senso, il metodo Monte Carlo ha un approccio estremamente semplice: risolve sistemi macroscopici attraverso la simulazione delle sue interazioni microscopiche.

La soluzione del problema attraverso l'utilizzo di questo metodo si ottiene campionando gli effetti delle interazioni microscopiche del sistema in esame tramite l'estrazione di numeri casuali, e ripetendo per un numero molto grande di volte questa operazione finché l'errore statistico non è inferiore alla precisione richiesta. Questo meccanismo di esecuzione richiede quindi la ripetizione di uno stesso calcolo, a volte anche molto semplice, per un gran numero di volte. Viste le caratteristiche di questo metodo di calcolo, diventa allora immediatamente chiaro perché sia indispensabile disporre di potenti calcolatori elettronici che siano in grado di effettuare in qualche frazione di secondo un gran numero di operazioni.

Il metodo Monte Carlo ha applicazioni in diversi campi quali la sociologia, economia aziendale, crescita della popolazione, finanza, genetica, dosimetria delle radiazioni e la radioterapia. Per realizzare un'applicazione del metodo Monte Carlo al calcolo della dose assorbita dal paziente per esposizione a un fascio di radiazioni, si deve effettuare un campionamento statistico dell'energia depositata dalle particelle del fascio attraverso il tracciamento del loro percorso; una volta che è nota la storia di ogni singola particella, la somma delle energie depositate da tutte le particelle componenti l'intero fascio è assunta come stima della dose assorbita. Come errore stati-

stico della stima si suole utilizzare la deviazione standard σ, il cui valore diminuisce tipicamente come:

$$S \propto \frac{1}{\sqrt{N}} \qquad (1.8)$$

nell'ipotesi, generalmente verificata, che la stima sia distribuita normalmente. A parte qualche eccezione, quindi, la precisione del metodo di calcolo cresce con l'aumentare del numero N di particelle processate durante la simulazione; per avere una stima sufficientemente precisa dell'energia depositata è necessario tipicamente seguire la storia di un numero di fotoni dell'ordine di $10^6 \div 10^8$: questo numero così grande pone seri limiti all'efficienza del metodo Monte Carlo. Nonostante l'introduzione di calcolatori elettronici sempre più veloci, è attualmente necessario attendere anche decine di ore prima di avere i dati completi di una simulazione; in queste condizioni diventa impossibile, per il radioterapista, avvalersi del metodo Monte Carlo per la compilazione del trattamento

1.4 Interazione delle radiazioni con la materia.

Per poter costruire la simulazione Monte Carlo descritta in precedenza è molto importante, come si intuisce subito, avere una conoscenza sufficientemente approfondita delle relazioni che descrivono il modo in cui le radiazioni interagiscono con il mezzo in cui si propagano. Bisogna innanzitutto dire che esistono diversi tipi di radiazioni e che il tipo di interazione che presentano dipende dalle caratteristiche delle radiazioni stesse e dalla loro energia; le radiazioni possono essere elettromagnetiche o corpuscolari: quelle elettromagnetiche includono i fotoni X e γ, mentre quelle corpuscolari includono le particelle α, gli elettroni e positroni β, gli elettroni di conversione interna, gli elettroni di Auger, i neutroni, i protoni e i frammenti di fissione; ai fini dell'argomento che voglio qui trattare considererò soltanto l'interazione dei fotoni con la materia.

Qualunque sia il metodo con il quale vengono prodotti (acceleratori lineari, radioisotopi o tubi a raggi), l'interazione dei fotoni con la materia dipende soltanto dalla loro energia e dalla densità e numero atomico del mezzo in cui si stanno propagando. A seconda dell'energia con la quale i fotoni urtano gli atomi del mezzo in cui si propagano si possono avere diverse categorie di interazioni, sia con gli elettroni che con i nucleoni dell'atomo; nell'intervallo energetico che intendo considerare nel mio lavoro, che va dalle centinaia di KeV alle decine di MeV, e per il numero atomico medio del mezzo in cui voglio studiare l'interazione dei fotoni, $Z \approx 7 \div 8$, sono però più probabili le seguenti tre interazioni: l'effetto Compton, l'effetto fotoelettrico e la produzione di coppie. Ciascuno dei tre effetti è preponderante in diversi intervalli energetici:

- Per energie basse ($0 \div 20$ KeV) prevale l'effetto fotoelettrico;
- Per energie medie (20 KeV $\div$ 20 MeV) prevale l'effetto Compton;
- Per energie alte (dai 20 MeV in su) prevale la produzione di coppie.

Il grafico riportato nella Figura 1.1 mostra quale è l'interazione più probabile, tra quelle che abbiamo descritto, in funzione dell'energia del fascio incidente e del numero atomico del materiale in cui si propaga; si può osservare come, nei materiali con $Z > 20$, l'effetto fotoelettrico sia preponderante per una energia del fotone compresa 0 e 100 KeV mentre la produzione di coppie predomina a energie superiori ai 10 MeV; nei materiali con $Z < 20$ l'interazione prevalente è invece l'effetto Compton

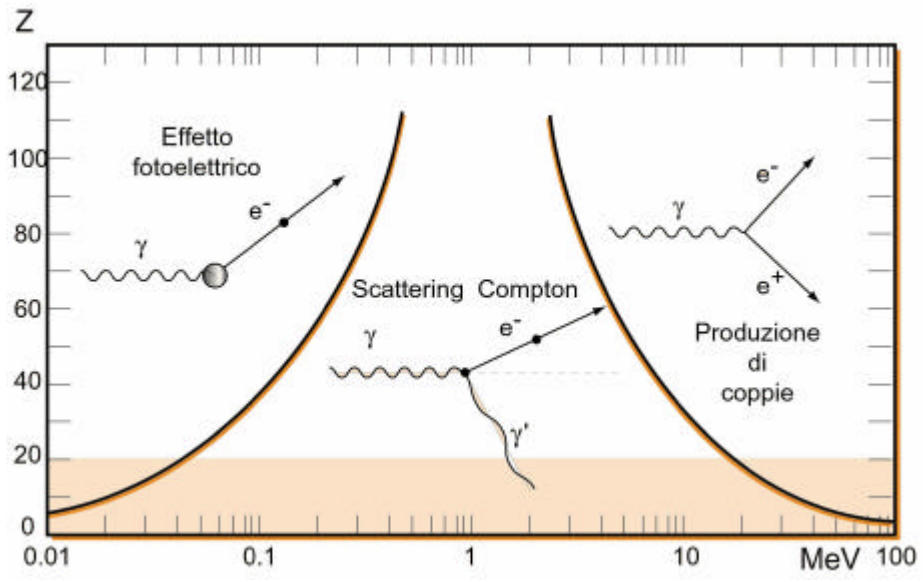

Figura 1.1

1.4.1 L'effetto fotoelettrico.

La fine del diciannovesimo secolo e l'inizio del ventesimo furono testimoni di una crisi negli studi della fisica: una serie di risultati sperimentali richiedeva, per essere spiegati teoricamente, l'accettazione di concetti totalmente incompatibili con la fisica classica; lo sviluppo di questi concetti, in un fascinante gioco di supposizioni radicali e brillanti esperimenti, condusse, alla fine, alla teoria quantistica. Nel 1900 Max Planck aveva già intuito che, per qualche ragione a lui sconosciuta, la radiazione di corpo nero veniva emessa in quanti di energia pari a $h\nu$, dove ν è la frequenza della radiazione emessa e h una costante, aggiunta dallo stesso Planck, pari a $6.63 \cdot 10^{-27}$ e rg·sec.

Un importante contributo all'accettazione del lavoro di Planck venne nel 1905 dal lavoro di Albert Einstein che utilizzò il concetto di natura quantistica della luce per spiegare alcune strane proprietà dei metalli che si presentavano quando questi sono irradiati con luce visibile e ultravioletta. Queste proprietà erano già state osservate da Hertz nel 1887, ma egli non riuscì a dare una spiegazione teorica soddisfacente in quanto aveva a disposizione solo conoscenze di fisica classica per dimostrare un fenomeno che poi Einstein dimostrò essere quantistico. Quello che si osservava era che:

1) placche di metallo lucido emettevano elettroni e non ioni positivi se irradiate;
2) se le placche emettessero o meno questi elettroni dipendeva dalla lunghezza d'onda della radiazione incidente; (effetto soglia)
3) l'intensità della corrente elettronica emessa era proporzionale all'intensità della sorgente luminosa;
4) l'energia dei fotoelettroni era indipendente dall'intensità della sorgente luminosa ma variava con la frequenza della luce incidente.

A partire da queste osservazioni e appoggiandosi ai risultati di Planck, Einstein riuscì a sviluppare una teoria che spiegava completamente l'effetto fotoelettrico considerando la radiazione incidente sul metallo composta da un insieme di quanti di energia $h\nu$: l'assorbimento di un singolo quanto da parte di un elettrone aumenta

l'energia dell'elettrone stesso di una quantità hν e questo, quando hν è maggiore di una certa energia di soglia, causa l'espulsione dell'elettrone dall'atomo, come rappresentato nella Figura **1.2**:

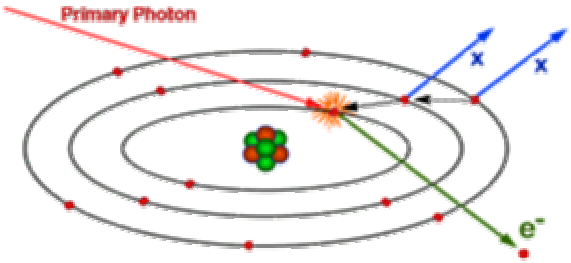

Figura 1.2

Una parte di questa energia quindi deve essere spesa per separare l'elettrone dal metallo; questa quantità, detta funzione di lavoro (W), ci si può aspettare che vari da metallo a metallo, ma è indipendente dall'energia dell'elettrone; il resto dell'energia che rimane all'elettrone viene trasformata in energia cinetica. Si ha quindi la seguente relazione tra la velocità v del fotoelettrone e la frequenza della radiazione incidente:

$$\frac{1}{2}mv^2 = h\nu - W \qquad (1.9)$$

L'effetto soglia e la relazione lineare tra energia cinetica e frequenza sono contenute in questa formula. Anche la proporzionalità tra la corrente elettronica e l'intensità della sorgente può essere capita in termini di quanti di luce: infatti una sorgente di luce più intensa emette più fotoni, e questi a loro volta possono liberare più elettroni.

Gli elettroni più fortemente legati dell'atomo hanno una probabilità maggiore di assorbire un fotone incidente rispetto a quelli delle shell più esterne; ad esempio, si ha che circa l'80% dei processi di assorbimento fotoelettrico avvengono con elettroni appartenenti alla shell K quando l'energia del fascio incidente è maggiore dell'energia di legame di questa shell; l'andamento della probabilità che avvenga l'effetto fotoelettrico in funzione dell'energia del fascio di fotoni incidente, rappresentato nel caso dell'Argento nella Figura 1.3, mostra dei massimi ben localizzati per energie pari alle energie di legame delle shell elettroniche più interne (K, L_I, L_{II}, L_{III}).

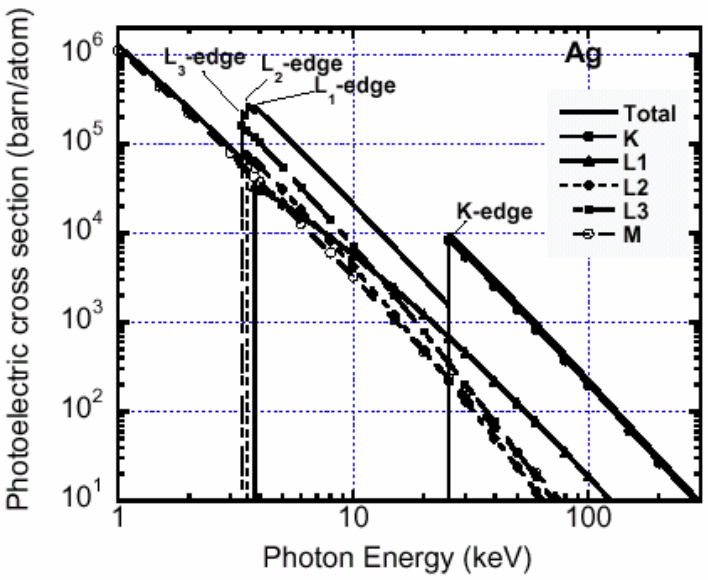

Figura 1.3

1.4.2 L'effetto Compton.

L'esperimento che diede la più diretta evidenza della natura particellare delle radiazioni è l'effetto Compton. Compton scoprì che le radiazioni di una certa lunghezza d'onda (compresa all'interno della regione dei raggi X) fatte passare attraverso un foglio metallico venivano scatterate con una distribuzione che non era consistente con la teoria classica delle radiazioni; secondo la teoria classica infatti il meccanismo dello scattering consisteva nell'emissione di radiazione luminosa da parte degli elettroni messi in oscillazione forzata dalla radiazione incidente: questo prediceva quindi una intensità luminosa osservata in un angolo q che variasse come $(1 + Cosq)$ e non dipendesse dalla lunghezza d'onda della radiazione incidente. Compton trovò che la radiazione scatterata con un certo angolo consisteva di due componenti: una la cui lunghezza d'onda era la stessa di quella della radiazione incidente e l'altra invece la cui lunghezza d'onda era spostata rispetto a quella della radiazione incidente di una quantità che dipendeva dall'angolo q come si vede nella Figura 1.4:

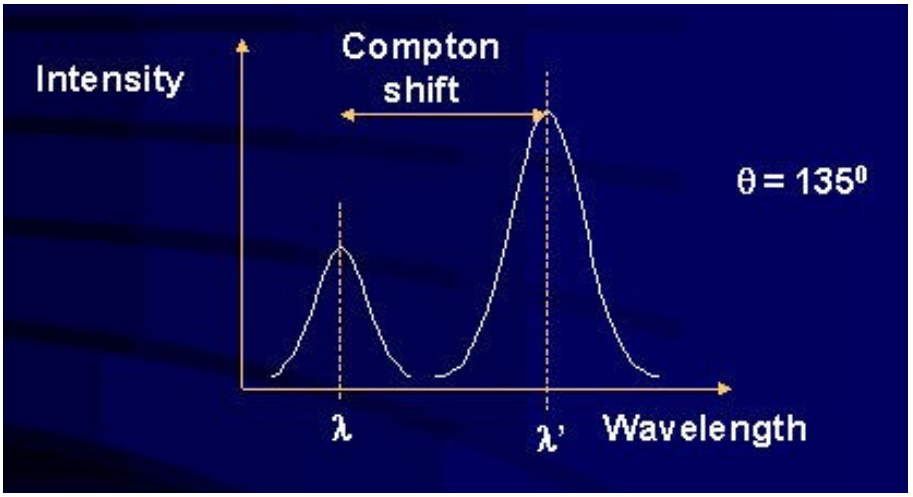

Figura 1.4

Compton fu in grado di spiegare la componente modificata della radiazione considerando la radiazione incidente come un fascio di fotoni di energia $h\nu$, dove h è la costante di Planck e vale $6.63 \cdot 10^{-27}$ erg·sec, in cui i singoli fotoni subivano scattering elastico con gli elettroni degli atomi componenti il metallo aventi una energia di legame molto inferiore all'energia del fotone incidente, come rappresentato nella Figura 1.5, e che quindi possiamo con buona approssimazione chiamare elettroni liberi.

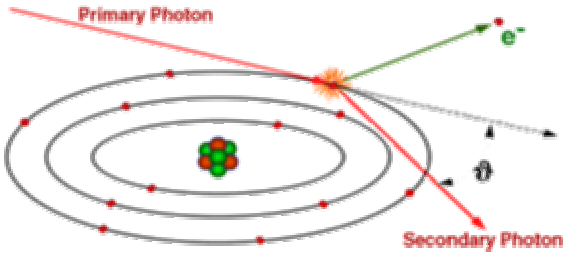

Figura 1.5

In un urto elastico devono essere conservati la quantità di moto e l'energia; per assegnare al fotone una quantità di moto, partiamo dall'equazione relativistica tra l'energia e la quantità di moto per una generica particella:

$$E = \left[\left(m_o c^2 \right)^2 + (pc)^2 \right]^{1/2} \qquad (1.10)$$

dove m_0 è la massa a riposo della particella; la velocità in funzione della quantità di moto è allora data da:

$$v = \frac{dE}{dp} = \frac{pc^2}{E} = \frac{pc^2}{\left(m_o c^2 + p^2 c^2\right)^{1/2}} \qquad (1.11)$$

La velocità di un fotone è però sempre uguale alla velocità della luce *c*, per cui la sua massa a riposo deve essere pari a zero; L'equazione relativistica dell'energia diventa perciò:

$$E = pc \qquad (1.12)$$

da cui si ricava facilmente che la quantità di moto del fotone è:

$$p = \frac{h\nu}{c} \qquad (1.13)$$

sostituendo $E = h\nu$.

Consideriamo ora un fotone con quantità di moto iniziale pari a $\bar{p}$ incidente su un elettrone a riposo; dopo l'urto, la quantità di moto del fotone è $\bar{p}'$ e l'elettrone rincula con una quantità di moto $\bar{\mathbf{P}}$; per la conservazione della quantità di moto si ha che:

$$\bar{p} = \bar{p}' + \bar{P} \qquad (1.14)$$

da cui segue che:

$$\bar{P}^2 = (\bar{p} - \bar{p}')^2 = \bar{p}^2 + \bar{p}'^2 - 2\bar{p}\cdot\bar{p}' \qquad (1.15)$$

La conservazione dell'energia porta invece all'equazione:

$$h\nu + mc^2 = h\nu' + \left(m^2 c^4 + P^2 c^2\right)^{1/2} \qquad (1.16)$$

dove *m* è la massa a riposo dell'elettrone; con qualche passaggio ricaviamo:

$$m^2 c^4 + P^2 c^2 = \left(h\nu - h\nu' + mc^2\right)^2 = (h\nu - h\nu')^2 + 2mc^2(h\nu - h\nu') + m^2 c^4 \qquad (1.17)$$

L'espressione di $\bar{\mathbf{P}}^2$ può ora essere riscritta come:

$$P^2 = \left(\frac{h\nu}{c}\right)^2 + \left(\frac{h\nu'}{c}\right)^2 - 2\frac{h\nu}{c}\frac{h\nu'}{c}\cos\theta \qquad (1.18)$$

da cui:

$$P^2c^2 = (h\nu - h\nu')^2 + 2(h\nu)(h\nu')(1-\cos\theta) \qquad (1.19)$$

dove θ è l'angolo di scattering. Sostituendo, ricaviamo infine:

$$h\nu\nu'(1-\cos\theta) = mc^2(\nu - \nu') \qquad (1.20)$$

o, equivalentemente:

$$\lambda' - \lambda = \frac{h}{mc}(1-\cos\theta) \qquad (1.21)$$

Le misure della componente modificata concordano molto bene con questa teoria; la componente non modificata è presumibilmente dovuta allo scattering del fotone con l'intero atomo: infatti, se al posto di m sostituiamo la massa dell'intero atomo, lo spostamento in lunghezza d'onda risulta molto piccolo in quanto un atomo è migliaia di volte più massivo di un elettrone.

La quantità h/mc ha le dimensioni di una lunghezza ed è chiamata "Lunghezza d'onda Compton dell'elettrone"; il suo valore è:

$$\frac{h}{mc} \cong 2.4\cdot 10^{-10}\,\text{cm} \qquad (1.22)$$

Sono state fatte anche misure del rinculo dell'elettrone e anche queste sono in accordo con la teoria. Successivamente è stato anche determinato, con una buona risoluzione temporale, da esperimenti di coincidenza che l'elettrone in uscita e il fotone rinculante appaiono simultaneamente.

1.4.3 La produzione di coppie.

Se l'energia di un fotone supera il doppio della massa a riposo di un elettrone, che è pari a 0.511 MeV, esiste una probabilità non nulla che, interagendo con il campo elettrico di un nucleo, il fotone scompaia dando luogo alla produzione simultanea di una coppia elettrone-positrone. Tutta l'energia del fotone in eccesso rispetto agli 1.02 MeV necessari alla produzione di queste coppie sono trasformati in energia cinetica che viene condivisa dall'elettrone e dal positrone. Poiché il positrone annichilerà in seguito al rallentamento nel mezzo in cui si propaga, sono normalmente prodotti due fotoni di annichilazione come prodotti secondari dell'interazione, come rappresentato nella Figura 1.6:

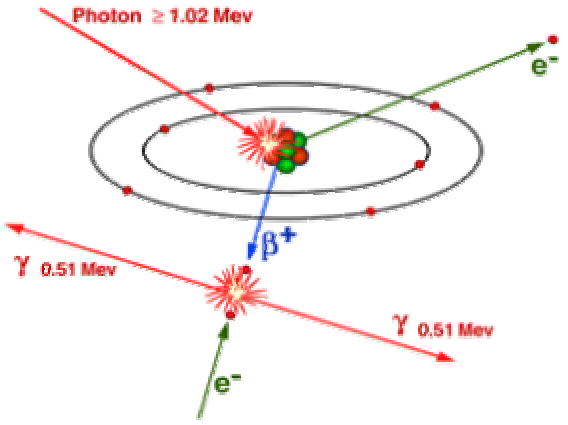

Figura 1.6

1.5 Attenuazione e assorbimento della radiazione elettromagnetica nella materia.

Le interazioni delle radiazioni elettromagnetiche con la materia sono dei fenomeni probabilistici: un fotone che attraversa un certo spessore "d" di materiale ha una certa probabilità "p" di attraversare il materiale e, ovviamente, una certa probabilità "1-p" di essere assorbito dal materiale. Analizzando il singolo fotone non saremo quindi in

grado di stabilire se esso sarà assorbito o no dal materiale in cui si propaga; esiste sempre, perciò, una probabilità, anche se molto piccola, che un fotone attraversi uno spessore di materiale grande a piacere. Se però consideriamo un fascio costituito da un gran numero di fotoni, ciascuno dei quali ha la probabilità "p" di passare attraverso il materiale di spessore "d", la frazione di fotoni che attraversa il materiale senza essere assorbito è uguale alla probabilità che ha ciascun fotone di attraversare il materiale.

Supponiamo ora di avere un fascio di fotoni di intensità I che attraversa uno spessore infinitesimo dx come rappresentato nella Figura 1.7:

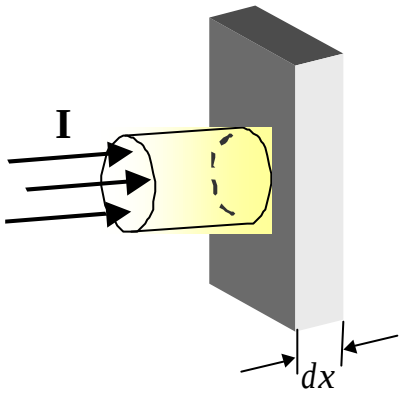

Figura 1.7

L'intensità del fascio emergente sarà ridotta di una quantità infinitesima dI che dipenderà dall'intensità del fascio incidente, dallo spessore dx e dal materiale di cui è fatto il materiale. Il parametro dipendente dal materiale è detto "coefficiente totale di attenuazione lineare" e si indica con la lettera μ. Si ha quindi che la riduzione di intensità del fascio incidente è data da:

$$dI = -I\mu\,dx \qquad (1.23)$$

dove il segno meno sta a indicare che c'é stata una riduzione di intensità. Se adesso supponiamo di avere tanti spessori dx uno di seguito all'altro, l'intensità dopo un generico spessore x si ottiene riscrivendo l'equazione precedente come:

$$\frac{dI}{I} = -\mu\,dx \qquad (1.24)$$

e integrandola tra I e I_o a sinistra e tra 0 e x a destra:

$$\int_{I_o}^{I} \frac{dI}{I} = \int_{0}^{x} -m\,dx \qquad \textbf{(1. 25)}$$

Risolvendo gli integrali si ottiene:

$$\mathrm{Ln}\left(\frac{I(x)}{I_o}\right) = -mx \qquad \textbf{(1. 26)}$$

che diventa, passando agli esponenziali:

$$I(x) = I_o e^{-m\cdot x} \qquad \textbf{(1. 27)}$$

L'intensità del fascio che ha attraversato il generico spessore x si è ridotta quindi del fattore $e^{-\mu\cdot x}$, il quale è uguale a zero solo per x = ∞; questa è la conferma matematica della osservazione fatta precedentemente, sulla base della natura probabilistica del fenomeno di assorbimento delle radiazioni elettromagnetiche. Il coefficiente μ ha le dimensioni dell'inverso di una lunghezza come si vede dal fatto che la funzione esponenziale è adimensionale; quindi, se x è espresso in cm, μ è espresso in cm^{-1}. Uno spessore x = 1/μ riduce l'intensità del fascio di un fattore 1/e ≈ 0.37, quindi del 37%.

Siccome la probabilità che un fotone attraversi un dato spessore x di un materiale senza subire alcuna interazione è data dal prodotto delle probabilità che il fotone non interagisca ne per effetto fotoelettrico che per effetto Compton che per produzione di coppie, si può osservare che il coefficiente totale di attenuazione lineare è dato dalla somma dei coefficienti di attenuazione lineare per questi tre singoli effetti:

$$m = t + s + k \qquad \textbf{(1. 28)}$$

dove il coefficiente di attenuazione per l'effetto fotoelettrico è dato da:

$$t = N\cdot t_a \qquad \textbf{(1. 29)}$$

dove N rappresenta il numero di atomi per centimetro cubo e τ_a la sezione d'urto atomica totale per l'effetto fotoelettrico, descritta nel paragrafo seguente; il coefficiente di attenuazione lineare per l'effetto Compton è invece dato da:

$$s = \mathrm{N} \cdot \mathrm{Z} \cdot s_{\mathrm{e}} = \mathrm{N} \cdot \mathrm{Z} \cdot \left(s_{\mathrm{A}} + s_{\mathrm{S}} \right) \qquad (1.30)$$

dove Z è il numero atomico e σ_e è la sezione d'urto totale Compton, descritta anch'essa nel paragrafo seguente, che è data dalla somma della sezione d'urto della radiazione scatterata Compton (σ_S) con la sezione d'urto di assorbimento (σ_A); il coefficiente di attenuazione lineare per la produzione di coppie è invece dato da:

$$k = \mathrm{N} \cdot k_{\mathrm{a}} \qquad (1.31)$$

dove κ_a è la sezione d'urto totale per la produzione di coppie, descritta anch'essa nel paragrafo seguente.

Come si può osservare, l'utilizzo del coefficiente di attenuazione lineare è limitato dal fatto che esso varia con la densità del mezzo in cui si propaga la radiazione, persino se questo mezzo è sempre lo stesso; per questo motivo, è molto più usato il coefficiente di attenuazione massico che è definito come:

$$m_{\mathrm{MASSICO}} = \frac{m}{r} \qquad (1.32)$$

dove ρ è la densità del mezzo. Per un dato fotone, il coefficiente di attenuazione massico non cambia con lo stato fisico del materiale assorbente: ad esempio, è lo stesso per l'acqua sia che essa si trovi nello stato liquido che nello stato gassoso. Nel caso di un composto, il coefficiente di attenuazione massico è dato dalla somma dei coefficienti di attenuazione massica dei singoli componenti ognuno moltiplicato per un opportuno fattore di peso.

Un'ultima grandezza che, come vedremo, sarà di fondamentale importanza nel calcolo della dose assorbita dal paziente, è il coefficiente totale di assorbimento lineare μ_a:

$$m_{\mathrm{a}} = s_{\mathrm{A}} + t + k \qquad (1.33)$$

che, come si può notare, è una quantità più piccola del coefficiente di attenuazione lineare μ.

1.6 Sezioni d'urto.

In prima approssimazione, la sezione d'urto può essere definita come la probabilità relativa che avvenga una certa reazione. Se consideriamo un rivelatore sistemato in maniera da rivelare delle particelle emesse in una direzione generica (θ,φ) rispetto alla direzione del fascio incidente, questo definisce un certo angolo solido $d\Omega$ rispetto al nucleo bersaglio. Sia ora la corrente del fascio incidente pari a I particelle per unità di tempo e sia il bersaglio costituito da N nuclei bersaglio per unità di superficie; se le particelle uscenti vengono rivelate con una frequenza pari a R, allora la sezione d'urto è definita da:

$$\sigma = \frac{R}{I \cdot N} \qquad (1.34)$$

Definita in questo modo σ ha le dimensioni di una superficie, ma può essere molto più piccola o molto più grande della superficie geometrica del disco del nucleo bersaglio visto dal fascio incidente; per un tipico nucleo di raggio R = 6 fm, ad esempio, la superficie geometrica πR^2 è circa pari a 100 fm^2 = 1 barn, ma per la cattura neutronica tramite ^{135}Xe la sezione d'urto è di circa 10^6 barn mentre per altre reazioni molto più improbabili la sezione d'urto può essere misurata in millibarn o in microbarn. Dunque σ ha le dimensioni di una superficie, e quest'ultima è proporzionale alla probabilità che avvenga una reazione.

Il rivelatore che abbiamo considerato occupa solo un piccolo angolo solido $d\Omega$ e perciò non riesce a osservare tutte le particelle uscenti dalla targhetta bersaglio; solo una piccola frazione dR è in questo modo rivelata, e perciò sarà misurata solo una frazione $d\sigma$ della sezione d'urto. Inoltre, le particelle uscenti non saranno generalmente emesse uniformemente in tutte le direzioni, ma avranno una distribuzione an-

golare che dipenderà da θ e, in alcuni casi, anche da φ. Se consideriamo questa funzione di distribuzione angolare rappresentata da r(θ,φ), allora:

$$dR = r(\theta,\varphi)\frac{d\Omega}{4\pi} \qquad (1.35)$$

dove il fattore 4π è stato introdotto per rendere il rapporto dΩ/4π una frazione pura. Possiamo quindi ora definire la quantità:

$$\frac{d\sigma}{d\Omega} = \frac{r(\theta,\varphi)}{4\pi \cdot I \cdot N} \qquad (1.36)$$

che è chiamata "sezione d'urto differenziale"; una misura di dσ/dΩ dà importanti informazioni sulla distribuzione angolare dei prodotti di reazione. È possibile ricavare la sezione d'urto totale σ dalla sezione d'urto differenziale dσ/dΩ integrando quest'ultima per θ che va da 0 a π e per φ che va da 0 a 2π:

$$\sigma = \int \frac{d\sigma}{d\Omega} = \int_0^{2\pi} \left(\int_0^{\pi} \frac{d\sigma}{d\Omega} \mathrm{Sen}\ \theta\, d\theta \right) d\varphi \qquad (1.37)$$

Vediamo ora quale è l'andamento delle sezioni d'urto per le interazioni descritte finora.

La probabilità che avvenga l'effetto fotoelettrico tra un fotone e un atomo interagenti è data dalla sezione d'urto atomica totale per l'effetto fotoelettrico:

$$\tau_a = k\frac{Z^m}{(h\nu)^n} \qquad (1.38)$$

dove k è una costante che dipende dall'energia del fotone incidente e dalla particolare shell considerata, Z è il numero atomico del materiale e m ed n sono delle quantità che dipendono sia da hν che da Z e che in prima approssimazione possiamo considerare come m = 4 ed n = 3. La distribuzione angolare dei fotoelettroni emessi è invece espressa, nel caso di radiazioni non polarizzate, dalla relazione:

$$\frac{d^2N}{d\Omega dV} \propto \frac{Sen^2 q}{\left[1-\frac{v}{c}Cosq\right]^4} \qquad (1.39)$$

dove θ è l'angolo tra la direzione del fotoelettrone emesso e quella del fotone incidente come rappresentato nella Figura 1.8:

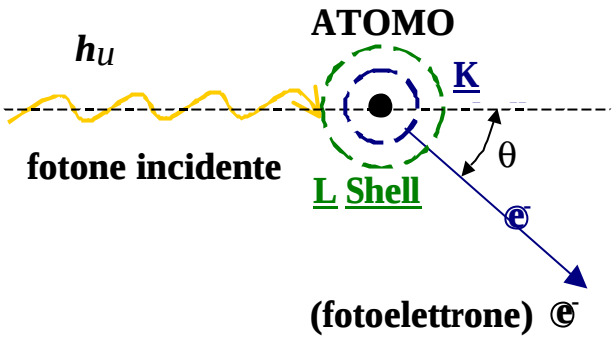

Figura 1.8

mentre nel caso di radiazioni polarizzate questa va moltiplicata per il fattore $Cos^2\phi$.

Per l'effetto Compton si può dimostrare che la probabilità che un fotone interagente con un elettrone libero di un atomo venga scatterato in una direzione θ, come rappresentato nella Figura 1.9, contenuta all'interno dell'angolo solido $d\Omega = 2\pi\,(Sen\theta\, d\theta)$ è data dalla sezione d'urto differenziale per lo scattering Compton:

$$\left[\frac{ds_e}{d\Omega}\right]_{K-N} = \frac{r_o^2}{2}\left\{1+\frac{a^2(1-Cosq)^2}{(1+Cos^2 q)[1+a(1-Cosq)]}\right\}\left\{\frac{1+Cos^2 q}{[1+a(1-Cosq)]^2}\right\} \qquad (1.40)$$

dove:

$$\begin{cases} r_o = \dfrac{e^2}{m_o c^2} = 2.818 \cdot 10^{-13}\ \text{cm} \\ a = \dfrac{E}{m_o c^2} \\ m_o c^2 = 511\ \text{KeV} \end{cases} \qquad (1.41)$$

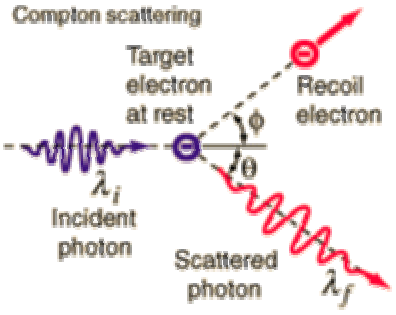

Figura 1.9

Questa formula è stata ricavata da Klein e Nishina facendo uso dell'elettrodinamica quantistica. Integrando la formula di Klein – Nishina per tutti i possibili valori di θ si ottiene la sezione d'urto totale Compton σ_e :

$$s_e = 2\pi r_0^2 \left\{ \frac{1+a}{a^2} \left[\frac{2(1+a)}{1+2a} - \frac{1}{a} \mathrm{Ln}(1+2a) \right] + \frac{1}{2a} \mathrm{Ln}(1+2a) - \frac{1+3a}{(1+2a)^2} \right\} \quad (1.42)$$

che rappresenta la probabilità che un fotone venga rimosso dal fascio incidente per effetto Compton per ogni singolo elettrone incontrato, cioè quando il fotone attraversa un materiale avente una densità elettronica pari ad (un elettrone)/cm^2. Nella Figura 1.10 possiamo vedere il grafico polare della sezione d'urto differenziale per lo scattering Compton in funzione dell'angolo di scattering θ per differenti valori dell'energia del fotone incidente.

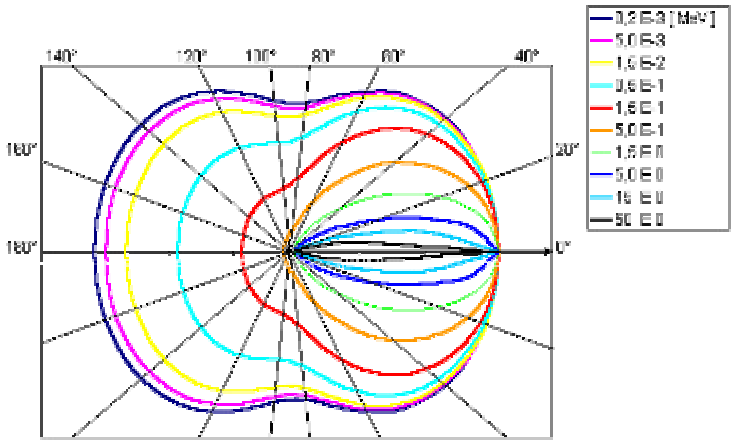

Figura 1.10

Per quanto riguarda la produzione di coppie, si può dimostrare che la sezione d'urto totale per atomo per la produzione di coppie è data da:

$$k_a = k_o Z^2 \left[\frac{28}{9} \cdot \mathrm{Ln}\left(\frac{2h\nu}{m_o c^2} - \frac{218}{27} \right) \right] \quad \textbf{(1. 43)}$$

dove:

$$k_o = \frac{1}{137}\left(\frac{e^2}{m_o c^2} \right)^2 = 5.80 \cdot 10^{-28} \frac{cm^2}{nucleo} \quad \textbf{(1. 44)}$$

Essa dipende quindi dal logaritmo dell'energia del fotone incidente.

1.7 Assorbimento dell'energia.

Un fascio di fotoni che si sta propagando in un mezzo interagisce con questo tramite gli effetti precedentemente descritti, producendo uno sciame di elettroni ed altri fotoni, detti secondari, diffusi in diverse direzioni. Questi elettroni, a loro volta, urtano gli atomi che incontrano e li ionizzano liberando ancora altri elettroni. Gli effetti della radiazione elettromagnetica sulla materia sono dovuti quasi esclusivamente a questi elettroni secondari. Essi possono avere quasi la stessa energia dei fotoni primari e perciò sono molto ionizzanti. La ionizzazione primaria degli atomi dovuta allo scattering Compton e all'effetto fotoelettrico dei fotoni è trascurabile rispetto all'ammontare della ionizzazione secondaria prodotta dagli elettroni secondari liberati in questi due processi dai fotoni. Per gli scopi pratici quindi, si considera che gli effetti dei fotoni sul mezzo attraversato siano quelli prodotti dagli elettroni secondari. Questi elettroni secondari interagiscono con la materia principalmente per:

- collisioni anelastiche con gli elettroni degli atomi;
- collisioni elastiche con i nuclei;
- frenamento (bremsstrahlung) da parte del campo elettrico dei nuclei.

Gli elettroni perdono la loro energia essenzialmente nelle collisioni anelastiche: essi infatti in questi urti cedono una piccolissima frazione di energia ai nuclei cambiando la direzione del loro moto, data la grande massa del nucleo rispetto a quella dell'elettrone, seguendo un cammino a zig-zag; dopo ripetute collisioni, l'effetto globale può essere quello di un backscattering dell'elettrone verso la direzione da cui era venuto. A causa di tutti questi urti, l'elettrone viene frenato e perde energia emettendo un fotone. Tuttavia la frazione di energia persa per bremsstrahlung è molto piccola per elettroni di energia inferiore ai 10 MeV, e diventa anche trascurabile per elettroni poco energetici (al di sotto di 1 MeV). Ad esempio un elettrone avente l'energia di 1 MeV perde in media l'1% della sua energia per bremsstrahlung e 32 eV per ogni atomo ionizzato: esso può quindi ionizzare circa 30000 atomi prima di perdere tutta la sua energia e di essere bloccato all'interno del mezzo che attraversa.

La perdita di energia degli elettroni nelle collisioni anelastiche con gli elettroni atomici è regolata dalla formula di Bethe-Bloch, ricavata da calcoli quanto-meccanici corretti. Essa esprime la perdida di energia per unità di lunghezza dE/dx (detta stopping power) di una particella carica che urta in modo anelastico con una particella identica (stessa massa):

$$-\frac{\mathrm{dE}}{\mathrm{dx}} = 2p\,\mathrm{N_a}\,\mathrm{r_0}^2\,\mathrm{m_0c}^2\,r\,\frac{\mathrm{Z}}{\mathrm{A}}\frac{1}{b^2}\left[\ln\frac{t^2(t+2)}{2(\mathrm{I}/\mathrm{m_0c}^2)^2} + \mathrm{F}(t) - d - 2\frac{\mathrm{C}}{\mathrm{Z}}\right] \quad \textbf{(1.45)}$$

dove:

$$F(t) = 1 - b^2 + \frac{\frac{t^2}{8} - (2r+1)\ln 2}{(t+1)^2} \quad \textbf{(1.46)}$$

r_0 è il raggio classico dell'elettrone, m_0 la massa dell'elettrone, N_a il Numero di Avogadro, I il potenziale medio di eccitazione, Z il numero atomico dell'elemento chimico di cui è fatto il materiale, A la massa atomica dell'elemento chimico di cui è fatto il materiale, r la densità del materiale, b il rapporto (v/c) tra la velocità dell'elettrone e quella della luce, d la correzione della densità, C la correzione di shell e t l'energia cinetica dell'elettrone incidente, in unità $m_0 c^2$.

Come si vede lo stopping power è proporzionale a Z: ecco perché gli elettroni sono bloccati in un breve cammino solo da materiali ad alto Z. Inoltre lo stopping power cresce logaritmicamente al crescere dell'energia cinetica degli elettroni.

Ciò significa che più gli elettroni sono energetici e maggiore è l'energia ad essi sottratta per unità di lunghezza negli urti anelastici con gli elettroni atomici. Il trasferimento di energia di un elettrone per ogni singola collisione è molto grande per cui la maggior parte dell'energia dell'elettrone avviene in poche collisioni. Il cammino percorso da un elettrone nella materia è molto piccolo perciò gli elettroni non sono molto penetranti rispetto ai fotoni che li hanno generati.

Nella Figura 1.11 è rappresentato il confronto tra la perdita di energia per unità di lunghezza di un fascio di elettroni e un fascio di protoni nel Rame in funzione dell'energia del fascio incidente.

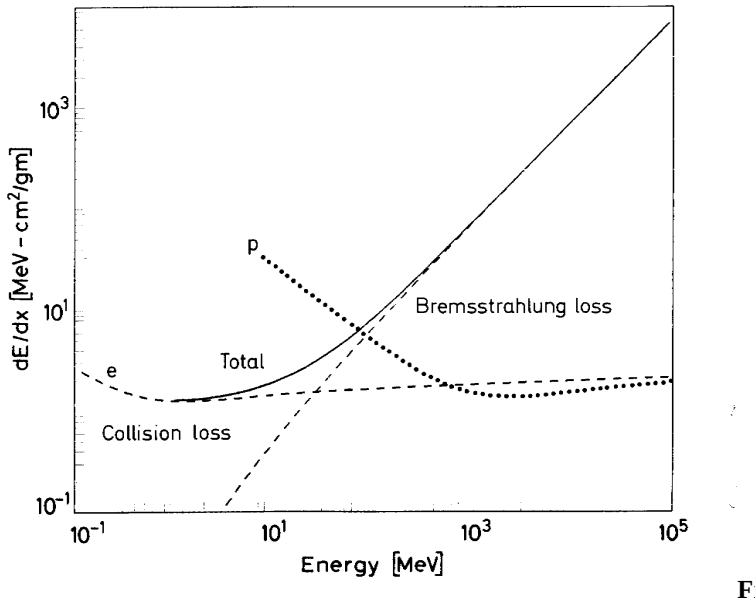

Figura 1.11

Il principio della radioterapia consiste nel trattamento di un tumore attraverso la ionizzazione generata dagli elettroni secondari; nella Figura 1.12 possiamo vedere una rappresentazione degli elettroni secondari prodotti da dei fotoni incidenti in un volumetto di materiale.

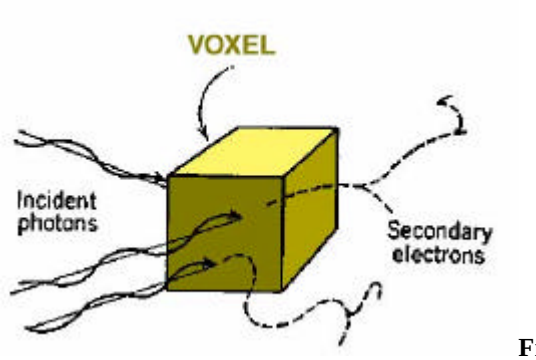

Figura 1.12

Tramite il piano di trattamento si fa in modo che la radiazione colpisca la zona del corpo in cui è presente il tumore da distruggere; tuttavia, con esso può venire distrutta anche una parte del tessuto sano che lo circonda: per questo motivo è importante per il radioterapista avere una descrizione, la più dettagliata possibile, della energia depositata in ogni elementino di volume nel quale si propaga la radiazione.

I fotoni sono molto più penetranti nella materia rispetto alle particelle cariche perché le sezioni d'urto dei tre processi di interazione dei fotoni con la materia sono molto più piccole della sezione d'urto delle collisioni anelastiche degli elettroni nel mezzo. Inoltre il fascio di fotoni primario è solo attenuato in intensità mentre attraversa il mezzo e non è degradato in energia: i fotoni che non hanno subito alcuna interazione (né assorbimento né scattering) passano dritti e conservano la loro energia. L'energia media assorbita da un elettrone secondario in una diffusione Compton di un fotone primario è $(h\nu)\sigma_A$; nell'effetto fotoelettrico tutta l'energia del fotone è ceduta all'atomo, una parte è utilizzata per estrarre il fotoelettrone e il resto è convertita in energia cinetica del fotoelettrone. L'energia cinetica del fotoelettrone è $(h\nu - B_e)$, dove B_e è l'energia di legame dell'elettrone atomico. Anche nella produzione di una coppia elettrone-positrone l'energia del fotone è interamente assorbita; l'energia cinetica totale della coppia elettrone-positrone è $(h\nu - 2m_oc^2)$. Trascurando i due termini B_e e $2m_oc^2$, la frazione di energia assorbita nello spessore dx del mezzo è:

$$dI = n(s_A + t + k)hn dx = nm_a hn dx = m_a I_0 dx \qquad (1.47)$$

La frazione:

$$\frac{dI}{dx} = m_a I_0 \quad \text{[in MeV/(cm}^3 \text{ s)]} \qquad \textbf{(1.48)}$$

dà il rate di energia assorbita per unità di volume nel mezzo irradiato dai fotoni, qualunque sia la dimensione e la forma dell'elemento di volume, supponendo che l'intensità I_0 del fascio irragiante sia costante. Dividendo questa quantità per la densità r del materiale si ottiene la dose assorbita o energia assorbita per grammo di materiale [in MeV/(g s)]:

$$R = \frac{m_a}{r} I_0 = \left(\frac{s_A}{r} + \frac{t}{r} + \frac{k}{r} \right) I_0 \qquad \textbf{(1.49)}$$

Fissata l'intensità I_0 del fascio incidente, la dose assorbita è indipendente dalla densità del materiale.

Si è detto che gli effetti e quindi anche i danni prodotti dalla radiazione sul corpo umano dipendono dal potere ionizzante della radiazione stessa. Quando la radiazione penetra nel corpo umano ionizza le molecole biologiche presenti nelle cellule, soprattutto il DNA o l'acqua, rompendo i legami chimici. I radicali che si formano attaccano altre molecole biologiche e danno luogo ad altre reazioni chimiche secondarie che "disfano" altre molecole. I processi biologici naturali tendono a riparare l'azione dannosa dei radicali, ma l'efficacia della loro azione sanante dipende dall'estensione del danno. Quindi può capitare che il danno provocato dalla radiazione sia completamente riparato oppure che non sia riparato: in quest'ultimo caso si può manifestare la morte della cellula intaccata, o l'alterazione genetica della cellula stessa, che viene trasmessa alle generazioni successive, o altri effetti secondari nel corpo del paziente e malattie, più o meno gravi. Nella cura di un tumore mediante radioterapia, l'effetto voluto con un opportuno irraggiamento del paziente, nel punto del corpo in cui è posizionato il tumore, è di uccidere le cellule del tumore per eliminarlo.

Capitolo 2

Il metodo Monte Carlo e la sua applicazione allo studio di sistemi fisici reali.

2.1 Introduzione.

Oltre ai tradizionali campi della fisica teorica e sperimentale, i progressi dell'elettronica e la possibilità di disporre di elaboratori sempre più potenti e veloci hanno aperto una terza via, che si colloca concettualmente tra le due: quella delle simulazioni numeriche. Con l'utilizzo dei calcolatori è infatti possibile creare un modello di un sistema fisico reale anche molto complesso e simularne l'evoluzione tramite un esperimento virtuale; l'esperimento così realizzato permette la misura delle grandezze fisiche proprie del sistema fornendo in questo modo delle risposte, che sono in ottimo accordo con la realtà, sulle modalità con cui il sistema stesso reagisce ai diversi stimoli esterni. L'evoluzione del sistema studiato dipende radicalmente dalla descrizione che ne viene fornita, cioè dalle equazioni che stanno alla base del modello; in particolare, possiamo fare una netta distinzione tra modelli deterministici e modelli stocastici:

- nei primi, ogni passo dell'evoluzione del sistema è determinato in modo univoco dalle condizioni presenti al passo precedente;
- nei secondi, invece, è il caso a giocare un ruolo fondamentale nel determinare le modalità di evoluzione del modello.

Il metodo Monte Carlo si può classificare come un esperimento virtuale nel quale si fa ricorso a strumenti statistici sia per realizzare l'evoluzione del sistema che per estrarne le successive informazioni. Il cuore di un programma basato sul metodo Monte Carlo è costituito da un meccanismo che sia in grado di simulare il verificarsi di eventi casuali: questo meccanismo è il generatore di numeri pseudo casuali; infatti, per simulare correttamente il comportamento di un sistema fisico la cui evoluzione è determinata dal caso, occorre disporre di uno strumento col quale generare sequenze casuali che abbiano una distribuzione di probabilità coincidente con quella delle variabili del sistema in oggetto.

2.2 Teoria della probabilità.

Per capire il metodo di simulazione Monte Carlo e interpretarne i risultati del calcolo è fondamentale introdurre qualche nozione elementare di teoria della probabilità. Consideriamo variabili casuali con distribuzione continua e studiamone la densità di probabilità in una e due dimensioni e la densità di probabilità cumulativa.

2.2.1 Densità di probabilità in una dimensione.

Una funzione di densità di probabilità $p(x)$ è una misura della probabilità di osservare l'evento x; per esempio, x potrebbe essere la posizione nella quale un fotone interagisce per effetto Compton. Se $p(x_1)=2\ p(x_2)$, allora questo vuol dire che un'osservazione di x in un intervallo differenziale dx contenente x_1 ha una probabilità

doppia di avvenire rispetto a una osservazione di x in un identico intervallo dx contenente invece x_2. Questo ovviamente ha significato solamente nel limite per dx che tende a zero o, se dx è finito, quando $p(x)$ varia di poco lungo l'intervallo nelle vicinanze di x_1 e x_2.

In generale, $p(x)$ deve soddisfare alcune proprietà speciali che la distinguono da tutte le altre funzioni:

- $p(x) \geq 0$, in quanto una probabilità negativa non ha significato;
- $p(x)$ è normalizzato nel seguente modo:

$$\int_{x_{min}}^{x_{max}} p(x)\,dx = 1 \qquad (2.1)$$

- $-\infty < x_{min} < x_{max} < +\infty$, cioè x_{min} e x_{max} possono essere numeri reali qualsiasi, con la sola condizione che x_{min} sia minore di x_{max}.

Queste sono le sole restrizioni su $p(x)$. Si può notare che la condizione di normalizzazione implica che $p(x)$ sia integrabile su tutto il suo intervallo di definizione; soddisfatta questa condizione, $p(x)$ può essere anche discontinua o addirittura infinita.

Certe funzioni di probabilità possono essere espresse in funzione dei loro momenti:

$$\langle x^n \rangle = \int_{x_{min}}^{x_{max}} x^n p(x)\,dx \qquad (2.2)$$

tuttavia, l'esistenza di questi momenti non è garantita o perfino necessaria. Assumendo che $\langle x \rangle$ e $\langle x^2 \rangle$ esistano, si può definire la varianza associata alla funzione probabilità come:

$$\mathrm{var}\{x\} = \langle x^2 \rangle - \langle x \rangle^2 \qquad (2.3)$$

La varianza è una misura dell'ampiezza della distribuzione di x: essa è uguale a zero solo per la funzione delta di Dirac mentre è maggiore di zero per tutte le altre funzioni di densità della probabilità, anche se queste sono combinazioni di funzioni delta.

Un esempio di grande importanza nella simulazione del trasporto degli elettroni è la funzione di densità di probabilità di Rutherford:

$$p(m) = \frac{a(2+a)}{2} \frac{1}{(1-m+a)^2} \quad ; \quad -1 \le m \le 1 \qquad (2.4)$$

dove m è il coseno dell'angolo di scattering, $\mathrm{Cos}(q)$; questa distribuzione assume, per piccoli valori dell'angolo q, la forma:

$$p(q) = 4a \frac{q}{\left(q^2 + 2a\right)^2} \quad ; \quad 0 \le q < \infty \qquad (2.5)$$

Il momento di ordine uno di questa distribuzione esiste ed è pari a:

$$\langle q \rangle = p\sqrt{\frac{a}{2}} \qquad (2.6)$$

ma il suo momento di ordine due non esiste; questo strano comportamento, che implica l'inesistenza della varianza angolare per questa distribuzione, è responsabile del fatto che è molto complesso studiare la traiettoria di un elettrone.

2.2.2 Densità di probabilità in due dimensioni.

La considerazione di distribuzioni a due o più dimensioni segue da una generalizzazione delle distribuzioni a una dimensione con l'aggiunta di una correlazione tra le osservabili e le loro probabilità; un esempio di una funzione di probabilità a due dimensioni *p(x,y)* è la distribuzione degli angoli e delle energie di un fotone che compie un urto anelastico con un atomo. Il significato di una densità di probabilità a due

dimensioni è il seguente: fissata una variabile, diciamo la *x*, la distribuzione risultante è una funzione di densità di probabilità nell'altra variabile *y*.

Le nozioni di normalizzazione e momenti seguono direttamente dal caso a una dimensione, per cui si ha:

$$\langle x^n y^m \rangle = \int_{y_{min}}^{y_{max}} \int_{x_{min}}^{x_{max}} x^n y^m p(x, y)\, dxdy \qquad (2.7)$$

con la condizione di normalizzazione:

$$\langle x^0 y^0 \rangle = 1 \qquad (2.8)$$

Quindi il momento di ordine zero è il solo che deve essere definito, mentre quelli di ordine più grande possono anche non esistere. Se esistono, è possibile allora definire la covarianza:

$$\mathrm{cov}\{x, y\} = \langle xy \rangle - \langle x \rangle \langle y \rangle \qquad (2.9)$$

che può essere positiva o negativa; si può notare che:

$$\mathrm{cov}\{x, x\} = \mathrm{var}\{x\} \qquad (2.10)$$

La covarianza è una misura dell'indipendenza stocastica delle variabili *x* e *y*; se *x* e *y* sono variabili casuali indipendenti, allora:

$$\begin{cases} p(x, y) = p_1(x) p_2(x) \\ \mathrm{cov}\{x, y\} = 0 \end{cases} \qquad (2.11)$$

ma non necessariamente è vero il viceversa.

Una funzione associata è il coefficiente di correlazione lineare o di Bravais – Pearson:

$$r\{x, y\} = \frac{\mathrm{cov}\{x, y\}}{\sqrt{\mathrm{var}\{x\} \mathrm{var}\{y\}}} \;;\; -1 \le r\{x, y\} \le +1 \qquad (2.12)$$

Esistono varie relazioni che riguardano la varianza e la covarianza, ad esempio:

$$\mathrm{var}\{x \pm y\} = \mathrm{var}\{x\} + \mathrm{var}\{y\} \pm 2\,\mathrm{cov}\{x, y\} \qquad (2.13)$$

che diventa più semplicemente:

$$\mathrm{var}\{x \pm y\} = \mathrm{var}\{x\} + \mathrm{var}\{y\} \qquad (2.14)$$

se *x* e *y* sono indipendenti.

La probabilità marginale di ogni variabile è definita integrando *p(x,y)* su tutto l'intervallo dell'altra variabile:

$$m(x) = \int_{y_{min}}^{y_{max}} p(x, y)\,dy \quad ; \quad m(y) = \int_{x_{min}}^{x_{max}} p(x, y)\,dx \qquad (2.15)$$

Si può notare che le densità di probabilità marginale sono correttamente normalizzate; ad esempio, nel caso della distribuzione dell'energia e dell'angolo di scattering, una delle distribuzioni di probabilità marginale è legata alla distribuzione delle energie indipendentemente della distribuzione degli angoli e l'altra è legata alla distribuzione degli angoli indipendentemente dall'energia. Quindi la funzione di distribuzione di probabilità può essere scritta come:

$$p(x, y) = m(x)\,p(y|x) \qquad (2.16)$$

dove la probabilità condizionale è definita da:

$$p(y|x) = \frac{p(x, y)}{m(x)} \qquad (2.17)$$

e rappresenta la probabilità che, per un dato valore di *x*, avvenga *y*. La presenza di m(x) nel denominatore garantisce la normalizzazione di *p(y|x)*.

2.2.3 Distribuzioni di probabilità cumulativa.

È possibile associare a ogni funzione di densità di probabilità a una dimensione la sua funzione di densità di probabilità cumulativa, che è definita come:

$$c(x) = \int_{x_{min}}^{x} p(x')\,dx' \qquad (2.18)$$

Le funzioni di densità di probabilità cumulativa soddisfano le seguenti proprietà che seguono direttamente dalla loro definizione e dalle proprietà delle funzioni di densità di probabilità:

- *p(x)* e *c(x)* sono legate tra loro dalla relazione:

$$p(x) = \frac{d}{dx}c(x) \qquad (2.19)$$

- *c(x)* vale zero all'inizio del suo intervallo di definizione:

$$c(x_{min}) = 0 \qquad (2.20)$$

- *c(x)* vale uno alla fine del suo intervallo di definizione:

$$c(x_{max}) = 1 \qquad (2.21)$$

- *c(x)* è una funzione monotona crescente di *x* come risulta dal fatto che *p(x)* è sempre positiva e dalla definizione di *c(x).*

Le funzioni di densità di probabilità cumulativa possono essere messe in relazione con una distribuzione uniforme di numeri casuali, in modo da fornire uno strumento per semplificare queste distribuzioni.

2.3 Generazione e bontà di una sequenza di numeri pseudo-casuale.

Abbiamo detto che il generatore di numeri pseudo casuali è il cuore della simulazione Monte Carlo. Si utilizza l'aggettivo pseudo casuale per classificare le sequenze di numeri, in genere distribuiti nell'intervallo [0,1), generate con tecniche deterministiche e riproducenti un comportamento casuale. Una gran quantità di tecniche sono state proposte per la generazione di numeri pseudo casuali, e tutte fanno uso principalmente di funzioni ricorsive tramite le quali si ottiene un nuovo numero della successione a partire dal precedente o, più in generale, da una combinazione dei precedenti, secondo uno schema del tipo:

$$x_{i+1} = f(x_0, x_1, \ldots, x_i) \quad (2.22)$$

dove x_0 è il valore iniziale, detto seme, a partire dal quale si calcola la sequenza x_1, x_2, …, x_i; quindi, essendo queste funzioni deterministiche, la successione x_1, x_2, …, x_i cambia al variare i x_0 ma, fissato questo, tutti i valori della sequenza sono a loro volta fissati.

Le tecniche utilizzate per la produzione dei generatori di numeri pseudo - casuali possono essere classificate in base alla bontà della successione prodotta. È importante, infatti, conoscere la bontà del generatore di numeri pseudo - casuali di cui si dispone: se la sequenza di numeri pseudo - casuali generati fosse non-uniforme, non varrebbero più tutte le leggi statistiche che si possono applicare alle distribuzioni ideali che seguono invece questo andamento; se il campione considerato deviasse da questa distribuzione uniforme, ad esempio, non varrebbe più il teorema del limite centrale di cui parleremo alla fine del capitolo. Le deviazioni di una distribuzione, da quella ideale alla quale ci si riferisce, sono sempre molto pericolose perché non è mai possibile avere una stima della perturbazione che questa può introdurre dai teoremi su cui normalmente ci si basa.

I parametri che distinguono tra loro i vari generatori sono: l'uniformità dei numeri pseudo casuali prodotti, la lunghezza del periodo delle sequenze di numeri generati,

la non uniformità dell'ordinamento interno delle singole sequenze, la non correlazione di una sequenza con le altre e l'efficienza del generatore. La qualità viene stabilita mediante una serie di test che sono stati messi a punto con l'obiettivo di stabilire la maggiore o minore casualità della sequenza prodotta.

2.4 Teoria del campionamento.

Ora che abbiamo visto gli elementi essenziali della teoria della probabilità elementare e della generazione di numeri pseudo casuali, possiamo adesso mettere assieme queste due nozioni e far vedere come i numeri casuali possono essere utilizzati per fare dei campionamenti a partire da delle distribuzioni di probabilità. Considereremo tre tipi di tecniche di campionamento: l'approccio diretto, il metodo del rigetto e un metodo misto che combina questi due.

2.4.1 Funzioni di densità cumulativa invertibili (metodo diretto).

Consideriamo una tipica funzione di densità di probabilità del tipo rappresentato in Figura 2.1:

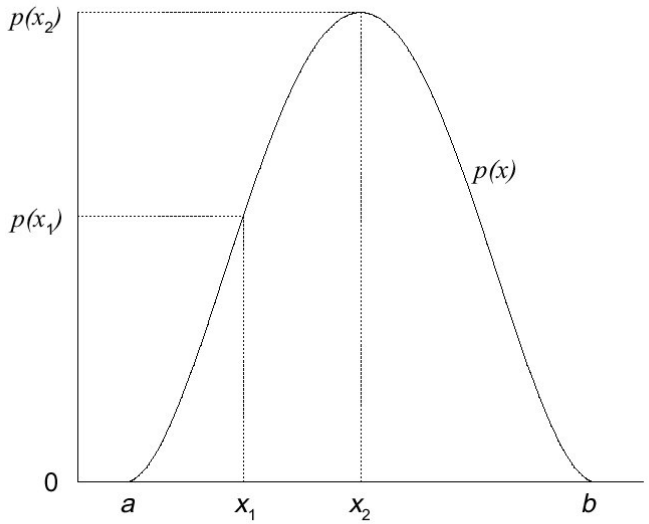

Figura 2.1

Questa è definita su un intervallo [a,b] dove né a né b devono essere necessariamente finiti. Una funzione di densità di probabilità deve avere le proprietà di essere integrabile (in modo che si possa normalizzarla integrandola sul suo intervallo di definizione) e non negativa (in quanto distribuzioni di probabilità negative sono difficili da interpretare). Costruiamo la sua funzione di densità di probabilità cumulativa:

$$c(x) = \int_a^x p(x')\,dx' \qquad \textbf{(2. 23)}$$

e assumiamo che sia propriamente normalizzata, cioè che *c(b)=1*. La corrispondente funzione di densità di probabilità cumulativa è rappresentata nella Figura 2.2:

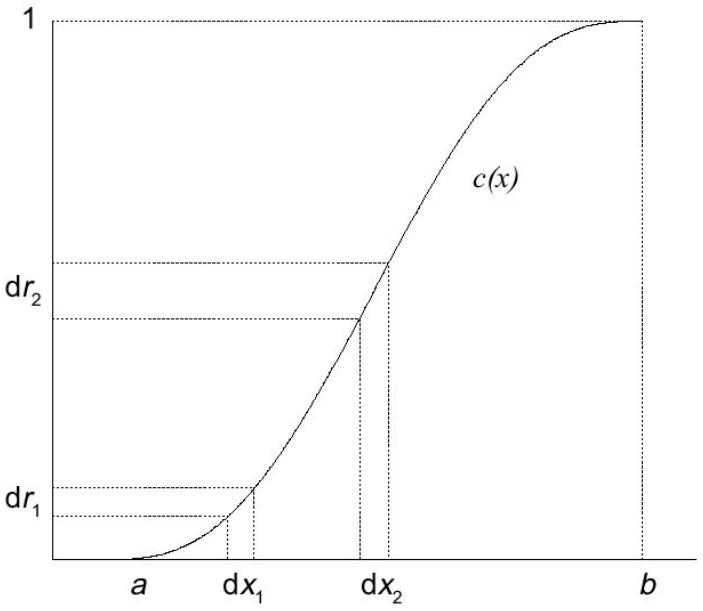

Figura 2.2

Per come è definita, è possibile mappare la funzione di densità di probabilità cumulativa sull'intervallo delle variabili casuali *r*, dove $0 \le r \le 1$ e *r* è distribuito uniformemente; quindi $r = c(x)$.

Ora, consideriamo due intervalli dx_1 e dx_2 di uguale ampiezza che siano degli intorni rispettivamente di x_1 e x_2; con qualche semplice calcolo si vede che:

$$\frac{dr_1}{dr_2} = \frac{\left[\frac{d}{dx}c(x)\right]_{x=x_1}}{\left[\frac{d}{dx}c(x)\right]_{x=x_2}} = \frac{p(x_1)}{p(x_2)} \qquad \textbf{(2. 24)}$$

Questo significa che, se si selezionano molte variabili casuali nell'intervallo [0,1], allora il numero che cade all'interno di dr_1 diviso il numero che cade dentro dr_2 è uguale al rapporto delle distribuzioni di probabilità in x_1 e in x_2.

Avendo mappato i numeri casuali sulla funzione di densità di probabilità cumulativa, possiamo invertire l'equazione per ottenere:

$$x = c^{-1}(r) \qquad (2.25)$$

Tutte le funzioni di densità di probabilità cumulativa che derivano da funzioni di densità di probabilità propriamente definite sono invertibili almeno numericamente se non lo sono analiticamente; quindi, scegliendo *r* casualmente su una distribuzione uniforme e sostituendola nell'equazione di sopra si genera *x* secondo l'appropriata funzione di densità di probabilità come si può vedere dalla Figura 2.3:

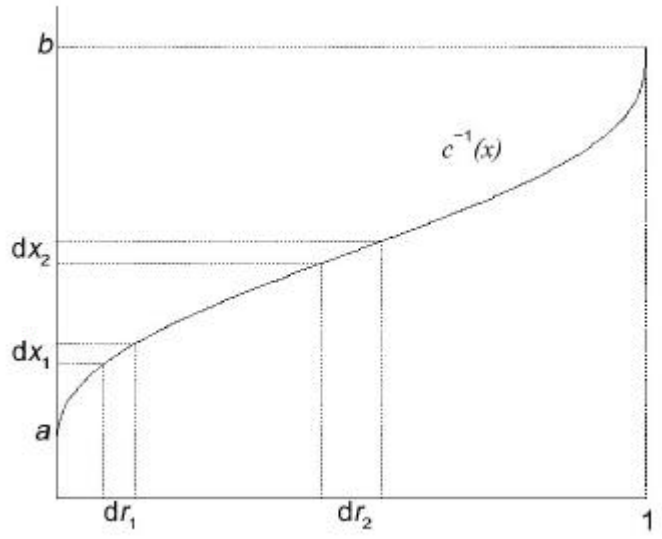

Figura 2.3

Consideriamo come esempio la funzione di densità di probabilità che governa la distanza *z* a cui avviene una interazione:

$$p(z)dz = m \cdot e^{-m \cdot z} dz \qquad (2.26)$$

dove μ è il coefficiente di interazione. L'intervallo valido di *z* è $0 \leq z < \infty$ e questa funzione di densità di probabilità è già propriamente normalizzata. La corrispondente funzione di densità di probabilità cumulativa e la sua mappa di numeri casuali è data da:

$$r = c(z) = 1 - e^{-m \cdot z} \qquad (2.27)$$

Invertendola si ha:

$$z = -\frac{1}{m}\ln(1-r) \qquad (2.28)$$

Se *r* è distribuita uniformemente nell'intervallo [0,1], allora questo è vero anche per *1 – r*. Una forma equivalente dell'equazione precedente è:

$$z = -\frac{1}{m}\ln(r) \qquad (2.29)$$

Questa è proprio la forma usata per calcolare la distanza di una particella da un punto di interazione in tutti codici Monte Carlo. È importante ricordare inoltre che, se il generatore di numeri casuali fornisce uno zero tra le sue possibilità, il campionamento indotto dall'equazione precedente causerà un errore di virgola mobile; è preferibile quindi avere un generatore di numeri casuali che non fornisca tra i suoi numeri lo zero esatto a meno che non serva per qualcosa di particolare.

2.4.2 Metodo del rigetto.

Mentre il metodo di inversione della funzione di densità di probabilità cumulativa è sempre possibile, almeno in principio, è spesso molto complicato calcolare $c(x)^{-1}$ sia a causa della complessità matematica della funzione che per il fatto che può contenere strutture matematiche difficili da controllare. Per ovviare a questo inconveniente si usa un metodo diverso, detto metodo del rigetto, la cui procedura è la seguente:

1) Si ridimensiona la funzione di densità di probabilità fino al suo massimo valore ottenendo una nuova funzione di distribuzione:

$$f(x) = \frac{p(x)}{p(x_{max})} \qquad (2.30)$$

che ha un valore massimo di uno in corrispondenza di x_{max} come si vede dalla Figura 2.4.

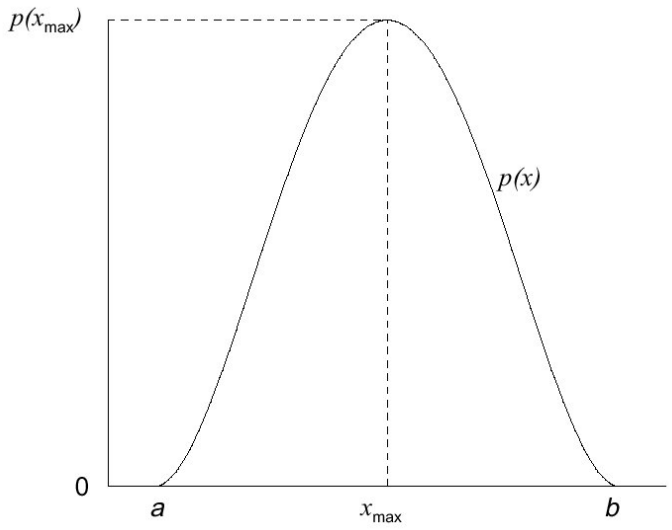

Figura 2.4

2) Si sceglie un numero casuale r_1 che abbia una distribuzione uniforme nell'intervallo [0,1] e lo si usa per ottenere una x che sia uniforme nell'intervallo [a,b] della funzione di densità della probabilità (per fare questo si deve calcolare $x = a + (b - a)\, r_1$). Questo metodo è ristretto a valori finiti di a e b; tuttavia, se sia a che b sono infiniti, è possibile trovare una trasformazione appropriata che permetta di lavorare in un intervallo finito (ad esempio, una $x \in [a,\infty)$ può essere trasformata in una $y \in [0,1)$ tramite una trasformazione del tipo $x = a\,[1 - \ln(1 - y)]$).

3) Si sceglie un secondo numero casuale r_2: se $r_2 < p(x) / p(x_{max})$ allora si accetta x, altrimenti si rigetta (regione ombreggiata sopra $p(x)/p(x_{max})$ nella Figura 2.5) e si ritorna al passaggio precedente.

Chiaramente, questo metodo funziona solo se la funzione di densità di probabilità è finita su tutto l'intervallo e se non è troppo complicato determinare la posizione del massimo. Se non fosse possibile determinare facilmente la posizione del massimo allora lo si può sovrastimare e il metodo funzionerà ancora correttamente ma con una efficienza minore.

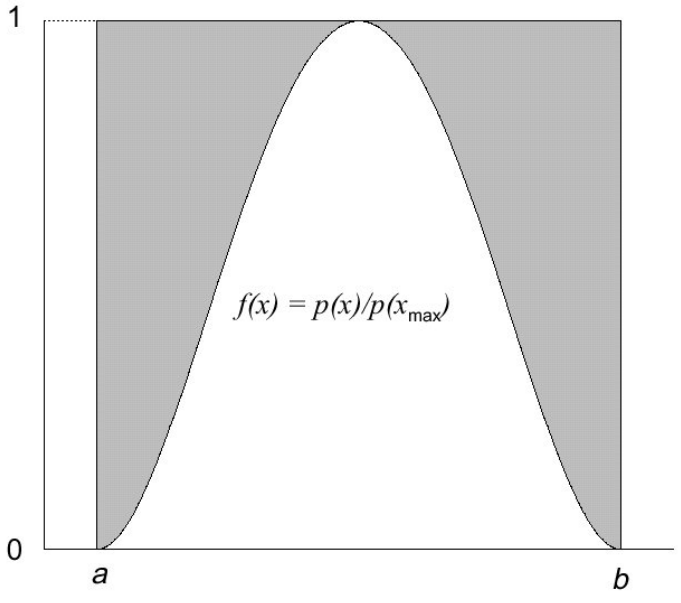

Figura 2.5

L'efficienza del metodo del rigetto è definita come:

$$\in = \frac{1}{p(x_{max})} \int_a^b p(x)\,dx \qquad (2.31)$$

quindi è data dal rapporto tra il numero previsto di coppie di numeri casuali che sono accettate e il numero totale di coppie che vengono utilizzate realmente.

2.4.3 Metodo misto.

Quando la funzione di densità di probabilità è troppo difficile da integrare e invertire e il metodo del rigetto è inefficiente, è possibile usare il metodo misto.

Supponiamo che la funzione di densità di probabilità possa essere fattorializzata come segue:

$$p(x) = f(x)g(x) \qquad (2.32)$$

dove *f(x)* è una funzione invertibile che contiene la maggior parte dei "picchi" mentre *g(x)* è relativamente piatta ma contiene la maggior parte della complessità matematica. Il metodo è il seguente:

1) Si normalizza *f(x)* generando una $\tilde{f}(x)$ tale che:

$$\int_a^b \tilde{f}(x)dx = 1 \qquad (2.33)$$

2) Si normalizza *g(x)* generando una $\tilde{g}(x)$ tale che:

$$\tilde{g}(x) \leq 1 \; ; \; \forall x \in [a,b] \qquad (2.34)$$

3) Usando il metodo diretto si sceglie una *x* utilizzando $\tilde{f}(x)$ come funzione di densità di probabilità;

4) Utilizzando questa *x* si applica la tecnica del rigetto usando $\tilde{g}(x)$, ovvero si estrae un numero casuale *r* uniformemente distribuito nell'intervallo [0,1] e se $\tilde{g}(x) \leq r$ allora si accetta la *x*, altrimenti si ripete l'operazione del punto precedente.

Tranne qualche rara eccezione, qualunque sia la complessità della funzione, è sempre possibile fattorializzare una funzione in questo modo.

Il metodo misto è anche equivalente a un cambio di variabili; sia:

$$p(x)dx = f(x)g(x)dx = \left(\tilde{f}(x)dx\right)\left(\int_a^b f(x)dx\right)g(x) \qquad (2.35)$$

dove $\tilde{f}(x)$ è ora una funzione di densità di probabilità propriamente normalizzata; utilizzando $\tilde{f}(x)$ come funzione per il metodo diretto, poniamo:

$$u = c(x) = \int_a^x \tilde{f}(x')dx' \; ; \; 0 \leq u \leq \int_a^b \tilde{f}(x')dx' = 1 \qquad (2.36)$$

come trasformazione tra *x* e *u*. Per definizione, l'inversa esiste ed è pari a $x = c^{-1}(u)$ e si ha $du = f(x)dx$. Perciò possiamo riscrivere l'equazione di $p(x)dx$ come:

$$p(x)dx = \left(\int_a^b f(x)\,dx \right) g(u)\,du \qquad (2.37)$$

dove è stata eliminata la $\tilde{f}(x)$ con un cambio di variabili. Quindi è possibile campionare *g(u)* utilizzando il metodo del rigetto (o qualche altra tecnica) e relazionare la *u* selezionata alla *x* attraverso la relazione inversa $x = c^{-1}(u)$.

Se il metodo del rigetto è utilizzato per *g(x)*, allora l'efficienza del metodo è data da una equazione identica a quella per il metodo del rigetto.

2.5 Calcolo dell'errore statistico.

Vediamo ora che errore si può associare ai risultati di una simulazione Monte Carlo. Abbiamo visto che questi sono prodotti tramite l'estrazione di un campione casuale, operata concretamente con l'utilizzo di un generatore di numeri casuali.

L'errore statistico associato alla stima è facilmente calcolabile quando è nota la sua distribuzione di probabilità; una distribuzione di variabile continua ricorrente ed estremamente importante è la distribuzione normale, o di Gauss, definita dalla funzione di densità:

$$p(x) = \frac{1}{s\sqrt{2p}} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2} \quad ; \quad -\infty < x < +\infty \qquad 2.38$$

dove *x* è la variabile casuale continua, *m* il suo valor medio e s^2 la sua varianza:

$$\begin{cases} m = \int_a^b x\ p(x)\ dx \\ s^2 = \int_a^b (x-m)^2 p(x)\ dx \end{cases} \qquad 2.39$$

L'importanza della distribuzione di Gauss risiede nel fatto che, come conseguenza del teorema del limite centrale, la somma di variabili continue comunque distribuite tende ad essere distribuita normalmente al crescere del numero di termini. Come caratteristica della distribuzione normale, si ha che il 99.74% della distribuzione è concentrata attorno al valor medio m in un intervallo pari a $[m + 3s, m - 3s]$; questo fatto, noto come la "regola del $3s$", ci consente di assumere s come errore della stima nel caso in cui siano verificate le ipotesi del teorema del limite centrale, cioè nel caso in cui il campione sia "sufficientemente numeroso".

Non sempre, però, questa condizione si può dire soddisfatta; per definire l'errore da associare alla variabile casuale in questione si fa ricorso, allora, a dei teoremi che garantiscono l'errore massimo entro il quale è sicuro che sarà contenuta la misura.

Un teorema fondamentale, in questi casi, è il teorema di Cebycev per il quale, data una variabile casuale x comunque distribuita, con un valor medio m e una varianza s^2, scelto un $\varepsilon > 0$ arbitrario allora la probabilità che lo scarto sia, in valore assoluto, maggiore di ε è minore di s^2/ε^2:

$$p(|x - m| \geq e) \leq \frac{s^2}{e^2} \qquad \textbf{2. 40}$$

Si suole indicare, in questo caso, una stima intervallare, ad un dato livello di confidenza (tipicamente almeno il 95%) del risultato ottenuto.

Questo teorema assicura, dunque, che deviazioni della variabile casuale x dal suo valor medio m, grandi rispetto alla deviazione standard, sono poco probabili.

Capitolo 3

Un software per il calcolo del tracciamento

3.1 Descrizione di un algoritmo per il tracciamento di un fascio di fotoni con il metodo Monte Carlo.

Vediamo ora come è possibile utilizzare le nozioni introdotte nei capitoli precedenti per applicare il metodo Monte Carlo al calcolo della traiettoria di ogni singolo fotone, componente un fascio di fotoni, che si sta propagando in un particolare mezzo che potremmo supporre essere l'acqua.

Abbiamo visto che nell'effetto Compton l'interazione avviene tra fotone ed elettrone: data l'energia $h\nu$ del fotone, la relazione di Klein – Nishina consente di calcolare la sezione d'urto del singolo elettrone. Se infatti definiamo:

$$r_o = \frac{e^2}{m_o c^2} = 2.818 \cdot 10^{-13}\ \text{cm} \qquad (3.1)$$

il raggio classico dell'elettrone e:

$$a = \frac{h\nu}{m_o c^2} \qquad (3.2)$$

dove $m_o c^2$ è la massa a riposo dell'elettrone, possiamo scrivere la sezione d'urto per effetto Compton per ogni singolo elettrone come:

$$\sigma_e = 2\pi r_o^2 \left\{ \frac{1+\alpha}{\alpha^2} \left[\frac{2(1+\alpha)}{1+2\alpha} - \frac{1}{\alpha} \ln(1+2\alpha) \right] + \right.$$
$$\left. + \frac{1}{2\alpha} \ln(1+2\alpha) - \frac{1+3\alpha}{(1+2\alpha)^2} \right\} \qquad (3.3)$$

La sezione d'urto complessiva sarà perciò ovviamente:

$$\sigma_A = Z \cdot \sigma_e \qquad (3.4)$$

dove Z è il numero atomico. Il coefficiente di attenuazione lineare per l'effetto Compton sarà quindi:

$$\mu_\sigma = \rho \frac{N_{AV}}{A} Z \cdot \sigma_e \qquad (3.5)$$

dove ρ è la densità del mezzo considerato in g/cm^3, N_{AV} è il numero di Avogadro e A è il numero di massa. Nel caso di composti come H_2O occorre considerare il peso molecolare M al posto del numero di massa e la somma dei numeri atomici al posto del singolo numero atomico; il coefficiente di attenuazione lineare per effetto Compton per l'acqua risulta perciò:

$$\mu_{\sigma\,(H_2O)} = \rho \frac{N_{AV}}{M} \sigma_e (Z_o + 2 \cdot Z_H) = 1 \frac{6.022 \cdot 10^{23}}{18} \sigma_e (8+2) =$$
$$= \left(3.345 \cdot 10^{23} \sigma_e\right) \text{cm}^{-1} \qquad (3.6)$$

Abbiamo visto inoltre che nel caso dell'effetto fotoelettrico è invece tutto l'atomo a partecipare all'evento; non sarebbe possibile infatti conservare l'energia e la quantità di moto se l'interazione fotoelettrica avvenisse con il solo elettrone.

La sezione d'urto per l'effetto fotoelettrico è in buona approssimazione direttamente proporzionale a Z^4 e inversamente proporzionale al cubo dell'energia del fotone incidente. Per calcolare il coefficiente di attenuazione lineare per l'effetto fotoelettrico utilizziamo una formula empirica:

$$m_t = \mathrm{k}\,\frac{\mathrm{Z}^4}{\mathrm{E}^3}\frac{r}{\mathrm{A}} \qquad (3.7)$$

dove l'energia del fotone E è espressa in *KeV*, k è una costante che vale circa:

$$\mathrm{k} = 24.2\,\frac{\mathrm{KeV}^3 \cdot \mathrm{cm}^2}{\mathrm{g}} \qquad (3.8)$$

Z è il valore efficace del numero atomico e il termine ρ/A rende conto del numero di atomi per unità di volume; nel caso dell'acqua si ha:

$$m_t = \left(24.2\,\frac{(7.5)^2}{\mathrm{E}^3}\frac{1}{18} \right)\mathrm{cm}^{-1} \qquad (3.9)$$

Il coefficiente di attenuazione lineare dovuto ad entrambi gli effetti (Compton e fotoelettrico) sarà dato quindi da:

$$m_{\mathrm{TOT}} = m_s + m_t \qquad (3.10)$$

nel caso in cui non consideriamo la produzione di coppie, la cui probabilità diventa trascurabile per fotoni di energie inferiori a 1.25 MeV.

Vediamo ora come usare il metodo Monte Carlo per stabilire la distanza percorsa da un fotone e il tipo di interazione a cui va incontro. Consideriamo la funzione di densità di probabilità cumulativa:

$$\mathrm{P}(\mathrm{x}) = 1 - \mathrm{e}^{-m \cdot \mathrm{x}} \qquad (3.11)$$

che, come abbiamo già visto, misura la probabilità che il fotone abbia un interazione nel tratto che va da $x = 0$ a x, dove abbiamo posto $\mu_{\mathrm{TOT}} = \mu$; ovviamente questa vale 0 per $x = 0$ e 1 per $x = \infty$. Per ricavare la distanza a cui un fotone avrà un interazione possiamo pensare allora di estrarre casualmente i valori di $P(x)$ nell'intervallo [0,1]

con una distribuzione uniforme e rovesciare l'equazione di sopra in modo da avere la distanza in funzione del numero casuale estratto che possiamo chiamare RND:

$$x = -\frac{\mathrm{Ln}(1-\mathrm{RND})}{m} \qquad (3.12)$$

Stabilita la distanza a cui avverrà l'interazione bisogna quindi decidere se l'interazione è di tipo Compton o fotoelettrico; per fare questo calcoliamo due valori m_σ ed m_τ tali che:

$$\begin{cases} m_s + m_t = 1 \\ \dfrac{m_s}{m_s} = \dfrac{m_t}{m_t} \end{cases} \qquad (3.13)$$

che rappresentano la probabilità che si abbia effetto Compton (m_σ) o effetto fotoelettrico (m_τ) proporzionalmente ai coefficienti di attenuazione lineare m_σ e m_τ; ovviamente, essendo complementari, è sufficiente calcolarne uno solo, ad esempio m_σ:

$$m_s = \frac{m_s}{m_s + m_t} \qquad (3.14)$$

A questo punto quindi si può estrarre un secondo numero casuale RND2 distribuito uniformemente tra 0 e 1 e controllare in quale dei due intervalli è caduto: se cade dentro m_σ il fotone scattera per effetto Compton altrimenti viene assorbito per effetto fotoelettrico:

$$\begin{cases} 0 \le \mathrm{RND2} \le m_s \Rightarrow \text{EFFETTO COMPTON} \\ m_s \le \mathrm{RND2} \le 1 \Rightarrow \text{EFFETTO FOTOELETTRICO} \end{cases}$$

Definito l'effetto, bisogna calcolare l'angolo con cui viene emesso l'elettrone nel caso dell'effetto fotoelettrico e le energie e gli angoli del fotone e dell'elettrone scatterati nel caso dell'effetto Compton; per quello che ci riguarda ci limiteremo al caso dell'effetto Compton.

Per quanto riguarda il calcolo dell'energia del fotone e dell'elettrone dopo lo scattering, queste sono facilmente ricavabili una volta noto l'angolo θ di scattering

del fotone rispetto alla direzione incidente; infatti si ha che, dalla relazione del Compton Shift e per la conservazione dell'energia:

$$\begin{cases} h\nu' = \dfrac{h\nu_0}{1+\alpha\cdot(1-\cos\theta)} & \Leftarrow \text{Energia del fotone scatterato} \\ T = h\nu_0 - h\nu' = h\nu_0 \dfrac{\alpha\cdot(1-\cos\theta)}{1+\alpha\cdot(1-\cos\theta)} & \Leftarrow \text{Energia cinetica dell'elettrone} \end{cases}$$

Il problema è invece quello di calcolare il valore dell'angolo θ di scattering estraendolo casualmente con la densità di probabilità della Klein – Nishina: infatti la Klein – Nishina non è una funzione invertibile analiticamente, quindi non è possibile usare il metodo diretto, e l'impiego del metodo del rigetto richiederebbe molto tempo per la sua esecuzione in quanto sarebbe molto alta la probabilità di dover rigettare i numeri estratti. Abbiamo quindi fatto ricorso al metodo numerico di bisezione per l'inversione della funzione.

Calcolato il nuovo valore dell'energia del fotone, si considera terminato lo step del fotone e si inizia il successivo seguendo nuovamente l'algoritmo descritto, fino a quando o il fotone ha raggiunto una energia inferiore alla soglia di interazione o finché non viene assorbito per effetto fotoelettrico; quando la storia di un fotone cessa di essere interessante ai fini del calcolo della dose ceduta al paziente a causa di una di queste due condizioni, inizia il processo di un altro fotone e così si va avanti finché non sono stati processati tutti i fotoni componenti il fascio.

3.2 Un modello di riferimento.

Il processo appena descritto è stato usato come base per lo sviluppo di una prima versione hardware dell'applicazione del metodo Monte Carlo; esso, infatti, per quanto notevolmente semplificato rispetto al modello realistico, può essere considerato sufficientemente rappresentativo da costituire un ottimo punto di partenza.

È stata pure realizzata una versione software dello stesso processo; questa è stata usata come metro di raffronto, sia per verificare la correttezza dei risultati che per appurare l'eventuale guadagno di efficienza della versione hardware.

Vista però l'estrema complessità delle relazioni fisiche che compaiono nell'algoritmo appena descritto si è preferito, ai fini di un più facile confronto tra la versione hardware e quella software, considerare un modello nel quale, alla vera densità di probabilità delle grandezze che appaiono, abbiamo sostituito una densità di probabilità uniforme; in questo modo, dal punto di vista software, ogni volta che il processo richiede l'estrazione di una variabile casuale, questa viene generata tramite la dichiarazione di una funzione di libreria tipica del linguaggio C che è la funzione "rand"; ogni volta che questa funzione viene richiamata dal programma, essa estrae un numero casuale, compreso tra 0 e un certo valore massimo RAND_MAX, con una distribuzione uniforme: dividendo il numero estratto per il valore di RAND_MAX si ottiene una distribuzione di numeri casuali uniformi compresi tra 0 e 1, che è proprio quella che abbiamo utilizzato nel nostro processo.

Dal punto di vista hardware, la realizzazione di questa distribuzione di numeri casuali uniformi è un problema non banale; alcune tecniche che consentono di ottenere delle distribuzioni di numeri pseudo-casuali verranno descritte in seguito.

Il modello da noi sviluppato dunque risulta molto semplificato dal punto di vista della statistica pur rispettando fedelmente la parte puramente meccanica del moto delle particelle; proprio grazie a questa semplificazione, però, questo sistema risulta molto facilmente studiabile ai fini del raffronto di prestazioni tra i due sistemi sviluppati: hardware e software.

3.3 Descrizione del programma.

Definiti i limiti del modello che abbiamo deciso di considerare, possiamo ora descrivere in che modo è stato possibile realizzare la simulazione del tracciamento di N particelle di un fascio che si propagano nella materia, con la relativa perdita di energia per ogni interazione subita, tramite il linguaggio C.

Nella Figura 3.1 è riportato l'algoritmo che è stato sviluppato:

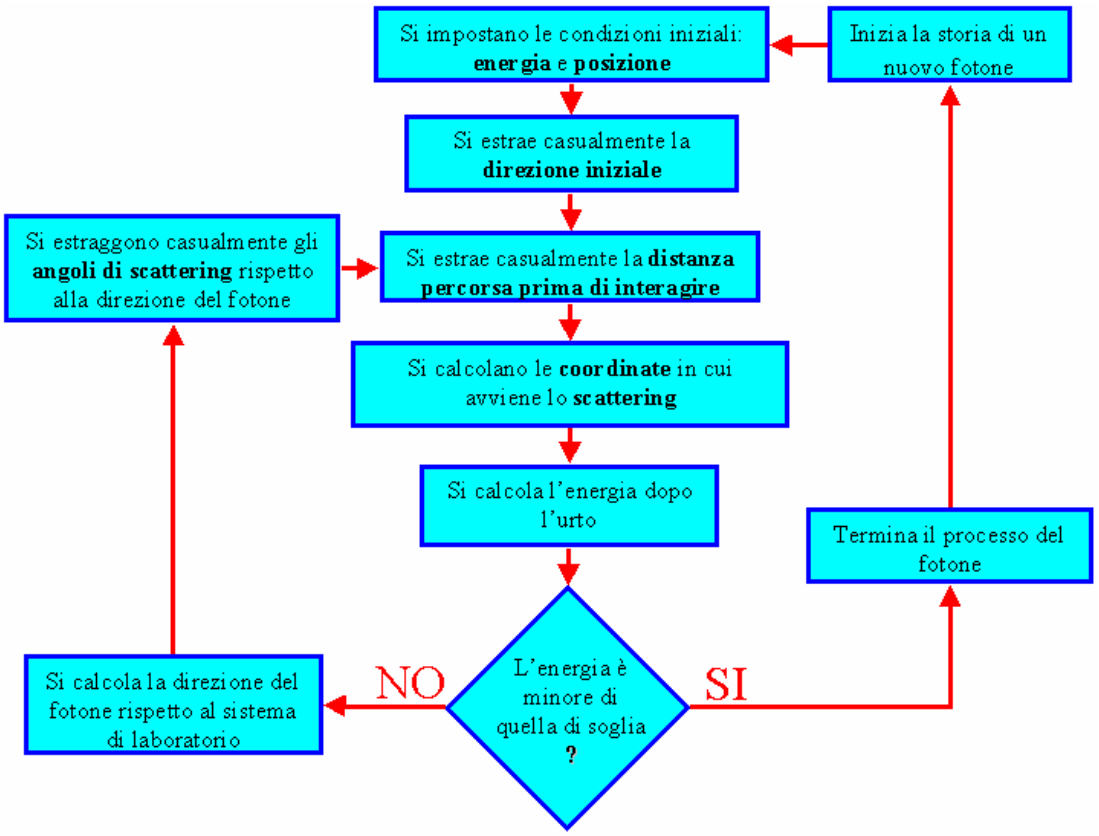

Figura 3.1

Come prima cosa, si definiscono l'energia e la posizione iniziale del primo fotone da processare: la prima l'abbiamo posta pari a 100 KeV, mentre la seconda coincide con l'origine del sistema di riferimento {0,0,0}; successivamente vengono estratti, chiamando per entrambi la funzione "rand" e moltiplicando il numero estratto rispettivamente per 2π e per π, gli angoli φ, tra la proiezione del vettore rappresentante la direzione del moto sul piano [XY] e l'asse X, e θ, tra il vettore rappresentante la direzione e l'asse Z, che definiscono la direzione del fotone all'inizio del processo. Tramite gli angoli φ e θ prodotti casualmente, sono allora calcolati i coseni direttori {u, v, w} del vettore direzione tramite le espressioni:

$$\begin{cases} u = \mathrm{Sen}(\theta)\ \mathrm{Cos}(\varphi) \\ v = \mathrm{Sen}(\theta)\ \mathrm{Sen}(\varphi) \\ w = \mathrm{Cos}(\theta) \end{cases} \qquad \textbf{3. 15}$$

A questo punto inizia il primo passo del fotone: viene chiamata la funzione "rand" e moltiplicato il numero prodotto per 7, in modo da estrarre casualmente la direzione percorsa dal fotone prima di interagire con il mezzo in un intervallo tra 0 e 7mm. Estratta la distanza d si effettua la traslazione del sistema di coordinate secondo le relazioni:

$$\begin{cases} x_{fin} = x_{in} + (x \cdot u) \\ y_{fin} = y_{in} + (x \cdot v) \\ z_{fin} = z_{in} + (x \cdot w) \end{cases} \qquad \textbf{3. 16}$$

e vengono estratti gli angoli di scattering Θ e Φ, relativi a un sistema di riferimento che ha l'asse Z parallelo alla direzione della particella prima dello scattering, dove Θ è l'angolo tra la nuova direzione del fotone (che poniamo coincidente con l'asse Z` di un nuovo sistema di riferimento) e quella precedente lo scattering, mentre Φ è l'angolo di cui è ruotato il piano di scattering $\pi_S$(Y`,Z`) attorno all'asse Z, come si vede nella Figura 3.2.

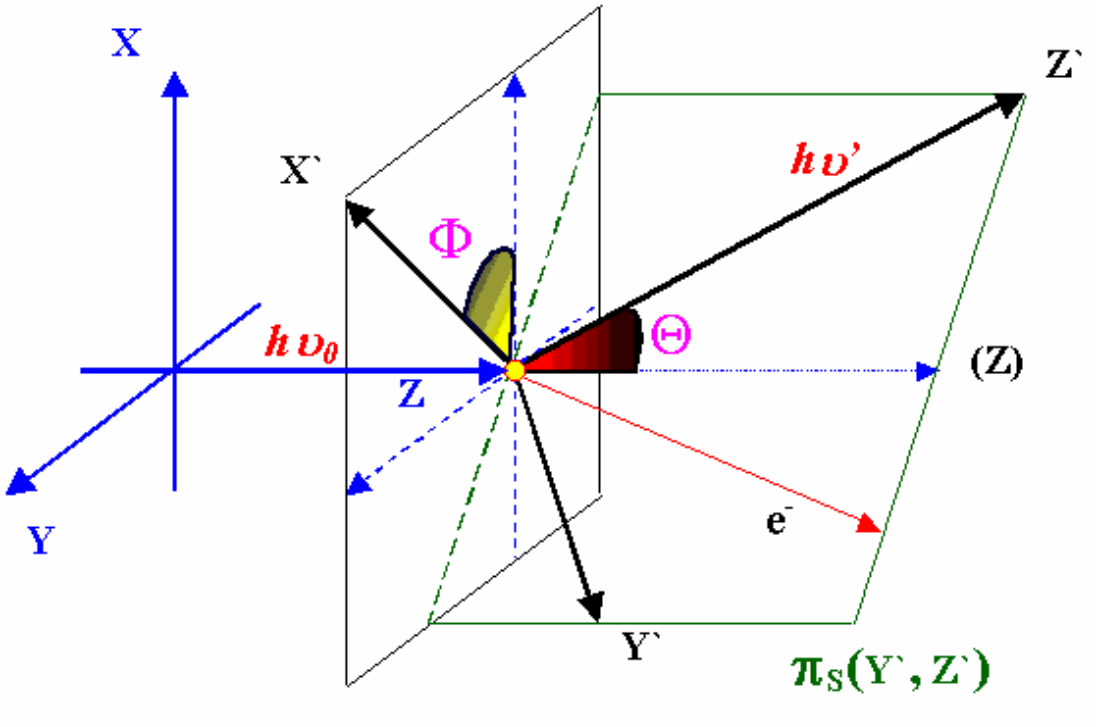

Figura 3.2

Si calcolano quindi i nuovi valori dei coseni direttori rispetto al sistema di riferimento fisso {X, Y, Z}, che possiamo chiamare sistema di laboratorio, secondo le relazioni:

$$\begin{cases} u = \mathrm{Sen}\,q\mathrm{Cos}\,j = u_o\mathrm{Cos}\,\Theta + \mathrm{Sen}\,\Theta(w_o\mathrm{Cos}\,\Phi\,\mathrm{Cos}\,j_o - \mathrm{Sen}\,\Phi\,\mathrm{Sen}\,j_o) \\ v = \mathrm{Sen}\,q\mathrm{Sen}\,j = v_o\mathrm{Cos}\,\Theta + \mathrm{Sen}\,\Theta(w_o\mathrm{Cos}\,\Phi\,\mathrm{Cos}\,j_o + \mathrm{Sen}\,\Phi\,\mathrm{Sen}\,j_o) \\ w = \mathrm{Cos}\,q = w_o\mathrm{Cos}\,\Theta - \mathrm{Sen}\,\Theta\,\mathrm{Sen}\,q_o\mathrm{Cos}\,\Phi \end{cases}$$

3. 17

dove con il pedice 0 abbiamo indicato i valori delle grandezze {u, v, w} e {θ, φ} prima dello scattering.

Dai nuovi valori di {u, v, w} si ricavano dunque i nuovi valori di $\mathrm{Sen}\theta_0$, $\mathrm{Sen}\varphi_0$ e $\mathrm{Cos}\varphi_0$, invertendo le relazioni 3. 17:

$$\begin{cases} \mathrm{Sen}\,(q_0) = 1 - w^2 \\ \mathrm{Sen}\,(j_0) = \dfrac{v}{\mathrm{Sen}\,(q)} \\ \mathrm{Cos}\,(j_0) = \dfrac{u}{\mathrm{Sen}\,(q)} \end{cases}$$

3. 18

i quali verranno utilizzati nello step successivo.

Infine calcoliamo l'energia persa dal fotone estraendo un numero casuale tramite la funzione "rand" e moltiplicandolo per 10, in modo che la massima energia persa sia di 10 KeV per ogni interazione, e la sottraiamo all'energia posseduta dal fotone prima dello scattering.

Questo processo si ripete finché la sua energia non scende al di sotto di una soglia di interazione che abbiamo prefissata pari a 1 KeV; una volta conclusasi la storia del primo fotone, l'intero algoritmo riparte dal principio e questo avviene tante volte quanti sono i fotoni che si intende simulare. Abbiamo riportato nella Figura 3.3 il risultato di una simulazione effettuata con questo algoritmo per un solo fotone e poi rappresentata graficamente tramite il software Mathematica™:

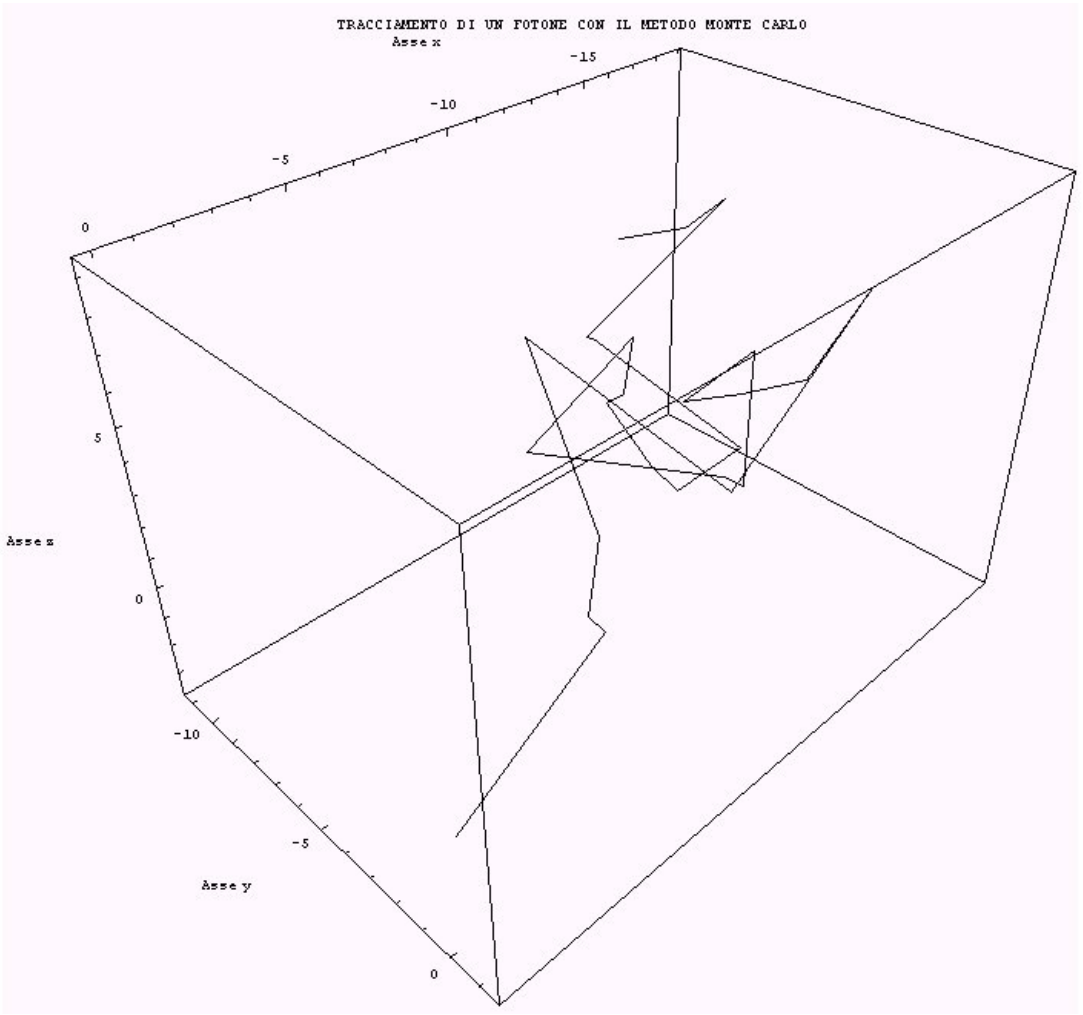

Figura 3.3

Da questo grafico sono evidenti i percorsi rettilinei effettuati dal fotone prima di subire uno scattering e la casualità del suo moto.

Capitolo 4

Field Programmable Gate Array (FPGA)

4.1 Descrizione di una FPGA

Le FPGA (Field Programmable Gate Array) sono dei circuiti integrati digitali programmabili dall'utente; questi circuiti sono stati commercializzati per la prima volta nel 1985 dalla XILINX con la produzione della famiglia XC2000 e possono essere visti come una evoluzione delle logiche programmabili ad alta densità come le PAL (Programmabile Array Logic) e le GAL (Generic Array Logic).

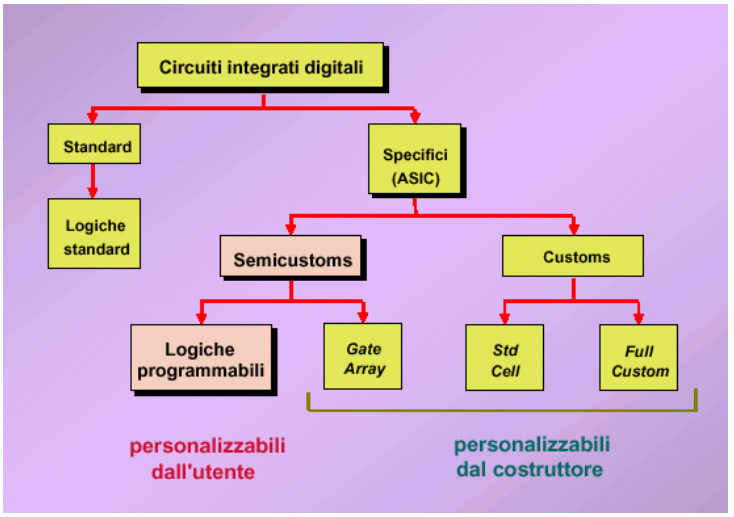

Figura 4. 1

Nella Figura 4. 1 possiamo vedere come le FPGA si inseriscono nella famiglia dei circuiti integrati digitali. Tali circuiti integrati sono basati su tecnologia "Static Ram", il cui elemento è la "Static Configuration Memory Cell". Queste celle RAM possono essere configurate caricando un "bit stream" che può essere letto direttamente da una PROM esterna, oppure essere scaricato da un PC tramite le linea seriale o parallela; la programmazione della FPGA può avvenire in alcuni millisecondi.

La cella è composta da due porte CMOS invertenti e un pass transistor che viene attivato per le operazioni di scrittura (programmazione della cella) o lettura (verifica della cella). Durante il normale funzionamento, la cella fornisce un continuo controllo all'elemento che indirizza e il pass transistor è aperto evitando eventuali perturbazioni al contenuto immagazzinato dalle porte CMOS (0 o 1 logico). Durante la programmazione, il pass transistor offre un percorso ai dati che devono entrare nel circuito di controllo di configurazione composto dalle porte CMOS. Durante la verifica il dato immagazzinato fluisce in direzione contraria.

L'architettura di una FPGA consiste di una matrice di macrocelle dette CLB (Configurable Logic Block) le quali comunicano fra loro e con dei dispositivi di I/O (Input/Output) attraverso una serie di canali verticali ed orizzontali chiamati routing channels. Nelle figure vengono mostrate le immagini della metallizzazione circuitale di un flip-flop (Figura 4. 2) e della FPGA (Figura 4. 3), ottenute al microscopio elettronico (ingrandimento di 3000X):

Figura 4. 2

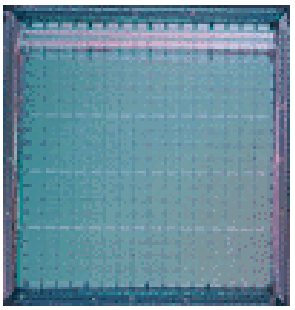

Figura 4. 3

Tale flip-flop contiene quattro celle di memoria i cui switch (parti colorate della figura) possono essere aperti o chiusi elettronicamente in modo abilitarne o meno la conduzione elettrica. Nella Figura 4. 4 è mostrata invece una sezione trasversale di una FPGA che mostra i 5 strati metallici di cui è costituita. Sfruttando

l'impostazione del dispositivo, è possibile selezionare in cascata le celle logiche di cui è costituito al fine di realizzare le funzioni logiche volute.

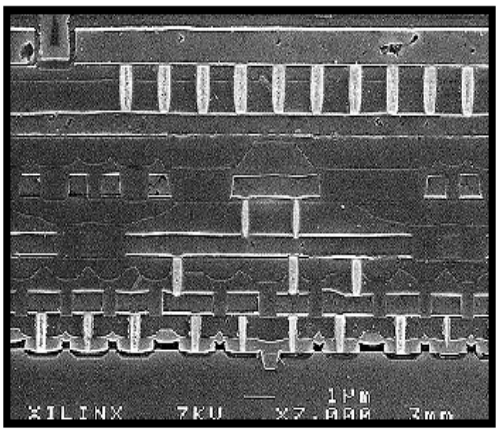

Figura 4. 4

Proprio questa loro duttilità ne ha consentito un rapido sviluppo e un utilizzo in campi che vanno dalla memorizzazione intensiva in registri, all'instradamento di dati, all'input-output veloce. Sono però soprattutto le applicazioni ad alte prestazioni il settore nel quale questo tipo di componenti trova maggior spazio, come l'interfacciamento a bus PCI con frequenze di 33MHz o 66 MHz, controllori DRAM basati su tempi dell'ordine dei ns, controllori DMA con ritardo clock-to-output sui 6 ns, e ancora applicazioni networking quali Ethernet, ATM e molte altre). Le caratteristiche principali delle FPGA sono:

- prestazioni elevate (Bus 200MHz, 1.8 volt LVDS (Low Voltage Differential Signaling));
- relativa facilità d'uso (descrizione con schematico o con un liguaggio HDL come VHDL, Verilog o Abel);
- possibilità della programmazione e riprogrammazione in-system.

Nella Figura 4.5 è mostrato lo schema generale di una FPGA XC4000 della XILINX; in tale figura si possono notare gli I/O Block, la matrice di CLB, le memorie interne ad ogni singola cella elementare LUTs (Look Up Tables), le interconnessioni tra le varie celle Switch Matrix e le interconnessioni programmabili (PI); tali elementi sono tutti configurabili dall'utente e permettono di realizzare la logica voluta.

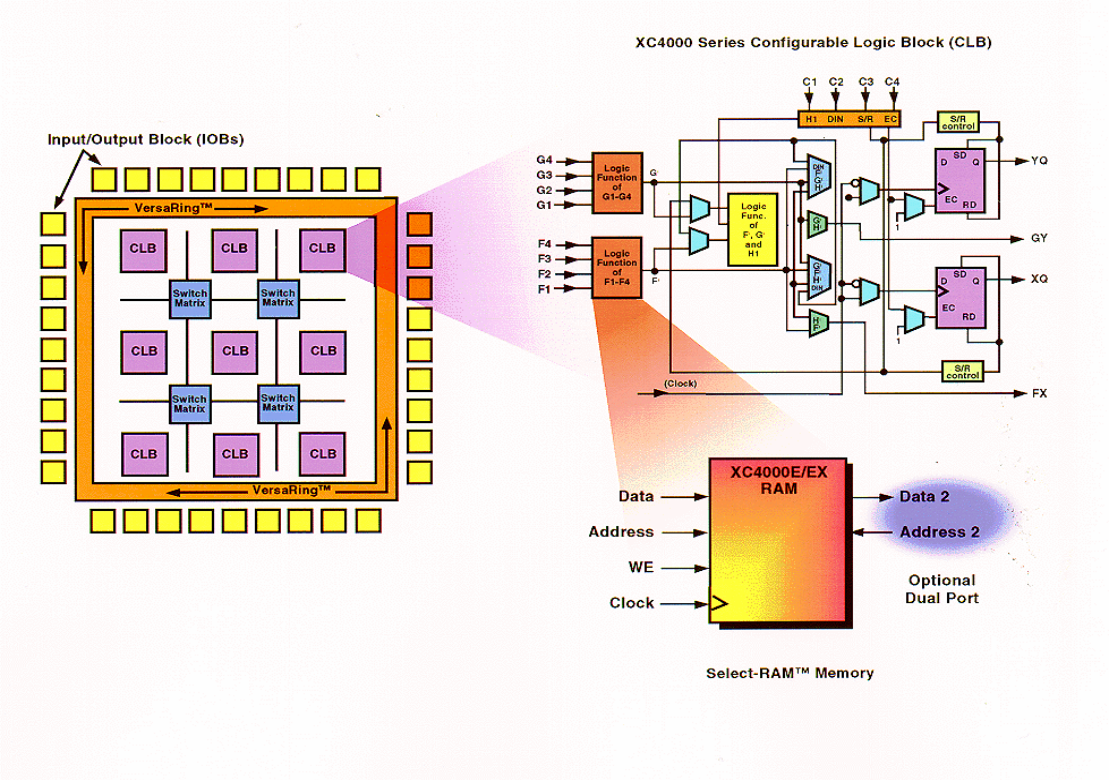

Figura 4.5

4.1.1 CLB.

I blocchi logici confugurabili (CLB) costituiscono gli elementi funzionali mediante i quali viene realizzato il progetto dell'utente all'interno della FPGA; essi hanno 13 ingressi {(G_1, G_2, G_3, G_4); (F_1, F_2, F_3, F_4); (C_1, C_2, C_3, C_4); CLOCK} e 4 uscite {(XQ, YQ); (X, Y)}. Dei 13 segnali in ingresso solo 8 sono configurabili; le funzioni logiche si possono implementare mediante le Look-up tables (LUTs) F e G, che possono generare funzioni a 4 ingressi e 1 uscita (F′ e G′); gli altri 5 ingressi arrivano dalle CLB circostanti (ad esempio H_1 è un ingresso del terzo generatore di funzioni H a 3 ingressi, e gli altri due ingressi possono essere F′ e G′, che sono le uscite di F e G; in questo modo le CLB sono collegate direttamente in cascata).

Nella Figura 4. 6 possiamo vedere l'architettura di una XC4000 della Xilinx:

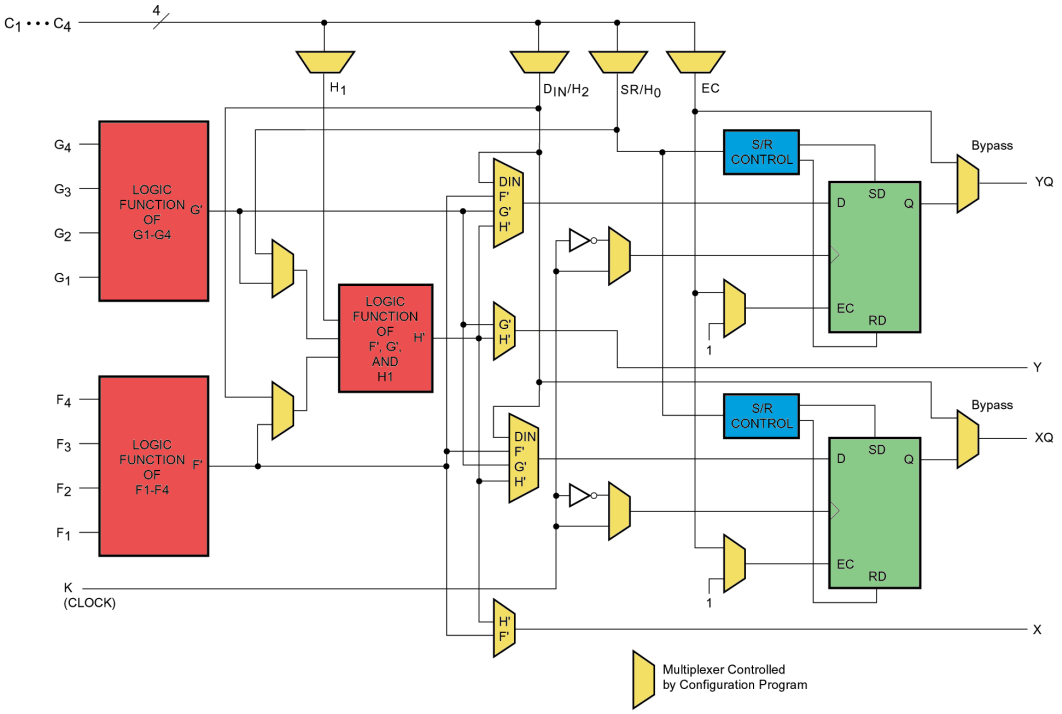

Figura 4. 6

In questa figura, X e Y rappresentano le uscite (esclusivamente combinatorie) del blocco: F′ o H′ possono essere connesse a X, mentre G′ o H′ possono essere connesse a Y. Complessivamente, quindi, in ciascun CLB possono essere realizzate o due funzioni indipendenti al più di quattro variabili, o una funzione di cinque, o una di quattro e una di cinque, oppure una funzione di nove ingressi. I due elementi di memoria sono dei flip-flop tipo D con segnali di clock (K) e di abilitazione (EC) comuni. Un terzo segnale comune (S/R) può svolgere, indipendentemente per i due registri, la funzione di set o di reset asincrono. I flip-flop possono essere triggerati sia sul fronte di salita che su quello di discesa e pilotano le uscite XQ e YQ; i dati in ingresso sono F, G′, H′ o DIN. I numerosi multiplexer che si notano in figura sono entità che, sulla base del bit stream di programmazione memorizzato nelle apposite celle SRAM, impongono alla CLB la configurazione definitiva, utile alla realizzazione del progetto globale impostato dall'utente.

4.1.2 Blocchi di I/O.

I blocchi di ingresso/uscita (IOB) realizzano l'interfaccia programmabile tra i piedini della FPGA (PAD) e le linee interne di segnale; tipicamente inglobano una circuiteria di protezione e di traslazione di livello verso standard interni a bassa dissipazione.

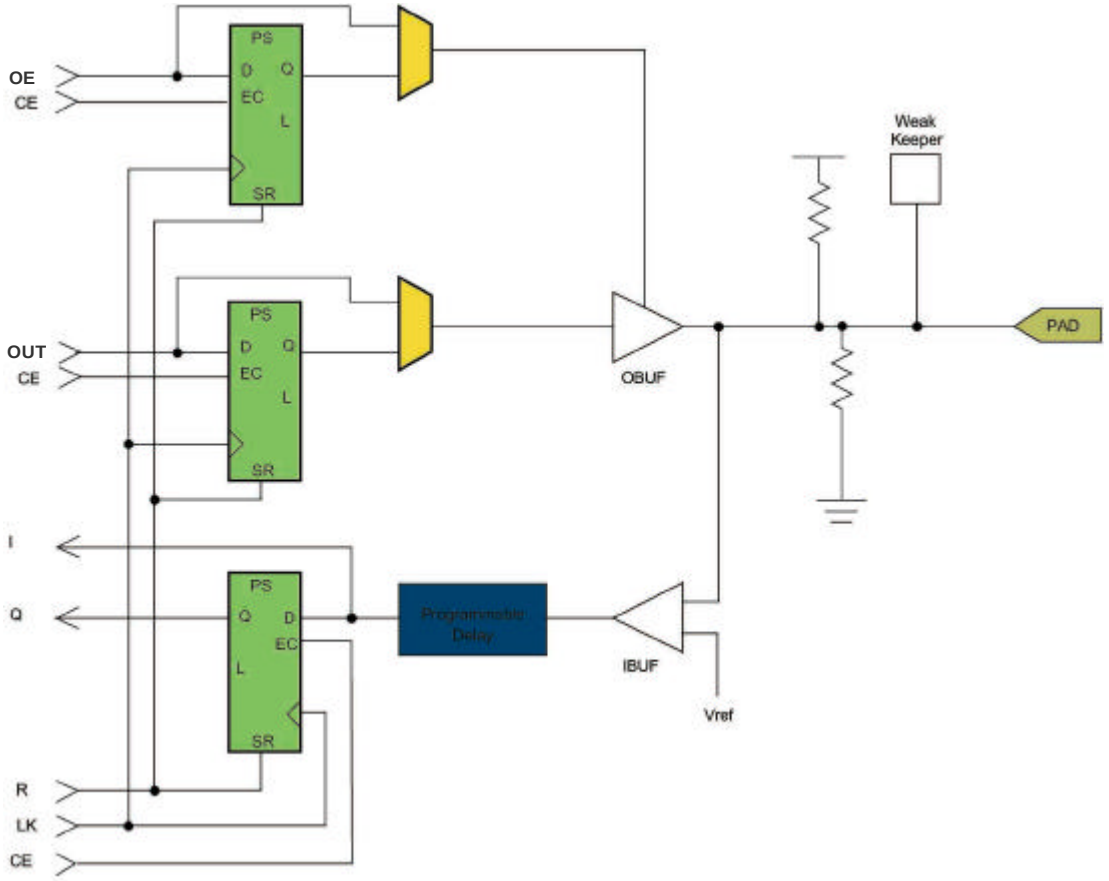

Figura 4.7

Ogni blocco rappresentato nella Figura 4.7 controlla un singolo piedino esterno, il quale può essere programmato come ingresso, uscita o bidirezionale. Il numero dei piedini di input/output della FPGA è compreso tra 84 e 800 circa e dipende dal tipo di package della stessa. I percorsi denominati I e Q conducono il segnale di ingresso, eventualmente sincronizzabile (input clock) in un latch o flip-flop, all'interno della matrice. Nel caso di utilizzo del registro di ingresso il dato proveniente dal pad può essere ritardato per compensare il ritardo del segnale di clock che deve passare attraverso un global buffer (con funzioni di Schmitt trigger) prima di arrivare all'IOB.

Anche per il segnale di uscita, i vari multiplexer di configurazione, presenti nello schema, assicurano diverse possibilità: il segnale di uscita OUT, eventualmente invertito, può essere memorizzato oppure giungere direttamente all'uscita. L'output buffer, pilotabile in alta impedenza da OE, realizza la bidirezionalità del pin o più semplicemente un'uscita a tre stati; resistori di pull-up e pull-down programmabili sono utili per forzare a Vcc o massa i piedini non utilizzati, in modo da ridurre i consumi.

4.1.3 Interconnessioni Programmabili e Switch Matrix.

Le interconnessioni programmabili (PI) permettono di definire gli opportuni percorsi conduttivi sia tra i vari input/output che tra i vari CLBs e IOBs. Tutte le connessioni interne alla matrice sono realizzate mediante segmenti conduttivi e punti di switch programmabili.

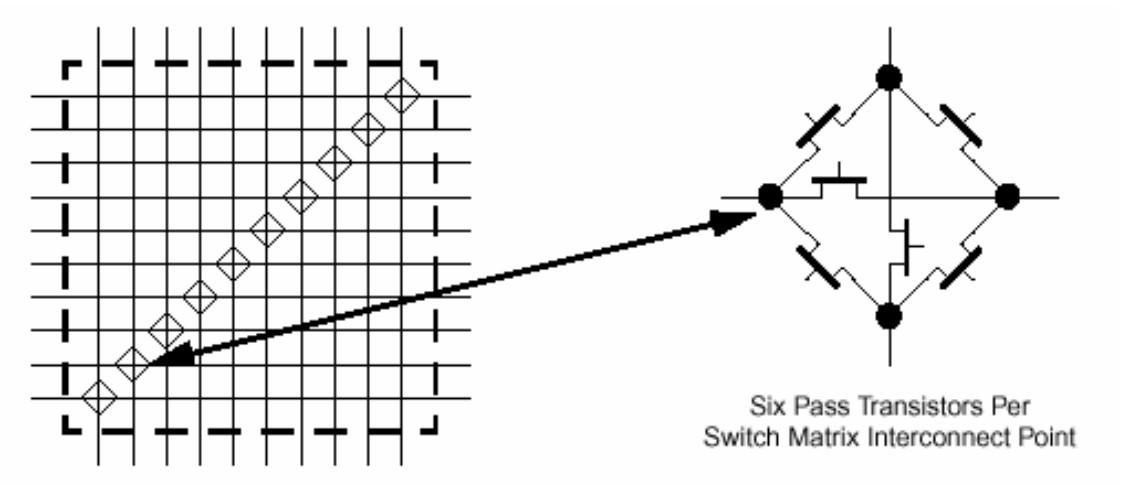

Figura 4.8

Ogni CLB è collegato con tutti i CLB della FPGA tramite delle interconnessioni e una matrice di switch programmabili (PSM). Nella Figura 4.8 possiamo vedere un vettore diagonale di punti di interconnessione.

L'intero sistema di collegamento è stato progettato con l'obiettivo di minimizzare la resistenza e la capacità del percorso medio tra elementi funzionali, migliorando così la qualità del dispositivo in termini di consumo e tempo di lavoro.

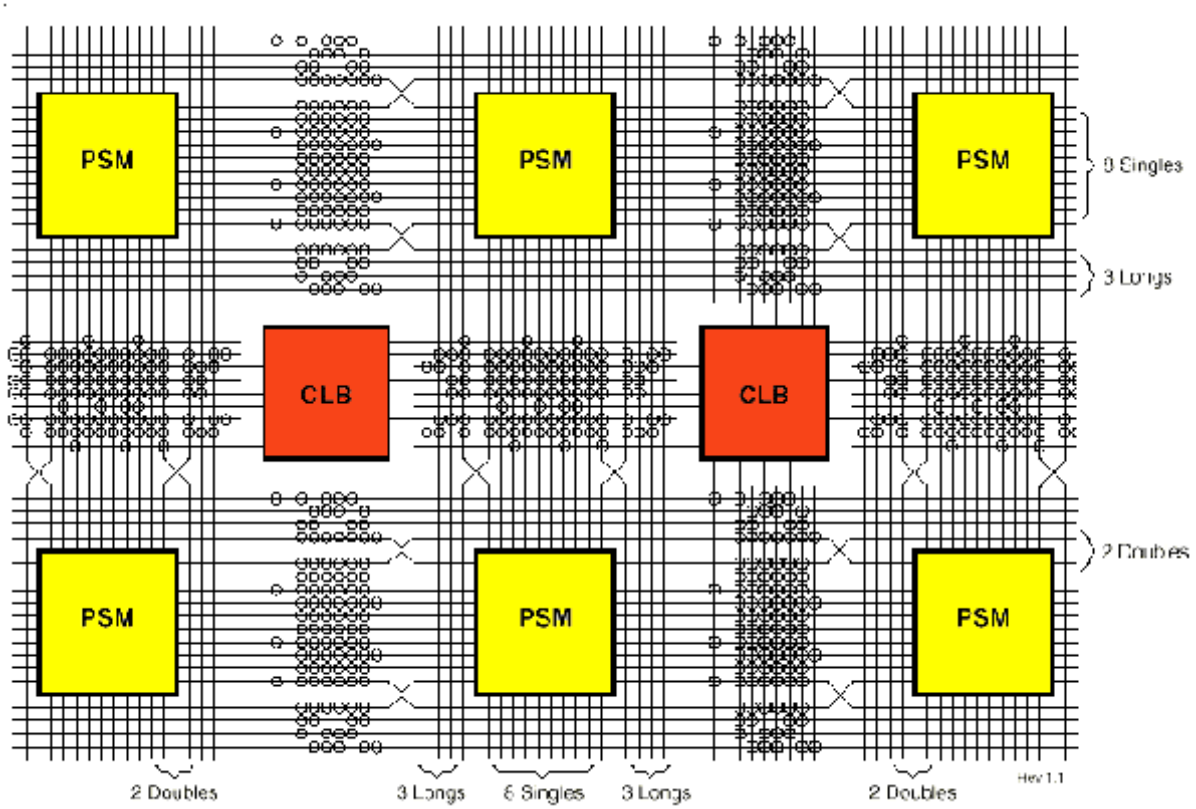

Figura 4.9

Esistono tre tipologie principali di interconnessione, rappresentate nella Figura 4.9, differenziate dalla lunghezza relativa dei segmenti conduttivi e dai tempi di propagazione: le "single-length lines", le "double-length lines", e le "longlines". Le single length lines costituiscono una griglia di linee verticali e orizzontali circondante ciascun CLB; ai vertici del blocco queste linee entrano in una switch matrix; in ogni switch matrix è presente un vettore diagonale di interconnect point, per ognuno dei quali si possono effettuare sei possibili collegamenti, grazie a sei transistor. In questo modo, una linea entrante da sinistra in una matrice può essere connessa ad un'altra linea entrante o da sopra, o da sotto, o da destra, o da tutte queste nel caso di connessione multipla. Le single length lines sono quindi utilizzate per condurre segnali in una area localizzata. Gli ingressi F1-F4, G1-G4 e C1-C4 del CLB possono essere pilotati da tutte le single-length lines adiacenti al blocco stesso, mentre K (clock) è

connesso solo alla metà di queste. Ogni uscita del CLB ha accesso a diverse single-length lines, sia orizzontali che verticali. Le double-length lines realizzano una griglia di segmenti conduttivi di lunghezza doppia rispetto alle single-length lines, nel senso che ciascuna linea "vede" lateralmente due CLB prima di rientrare in una switch matrix. La rete di interconnessione definita dalle longlines spazia invece lungo tutta l'area della matrice.

Le connessioni fra i vari CLB ed i blocchi IOB sono garantite invece da canali verticali ed orizzontali, come si può vedere nella Figura 4.10; in particolar modo i blocchi di I/O formano un anello attorno alla matrice di blocchi configurabili (Versa Ring), mentre le intersezioni dei canali sono gestite da una matrice programmabile di smistamento (PSM, Programmable Switch Matrix), la quale consente la selezione di connessioni a singola, doppia e lunga linea. In tal modo ogni segnale in ingresso nella FPGA può essere propagato su tutta la matrice di CLB.

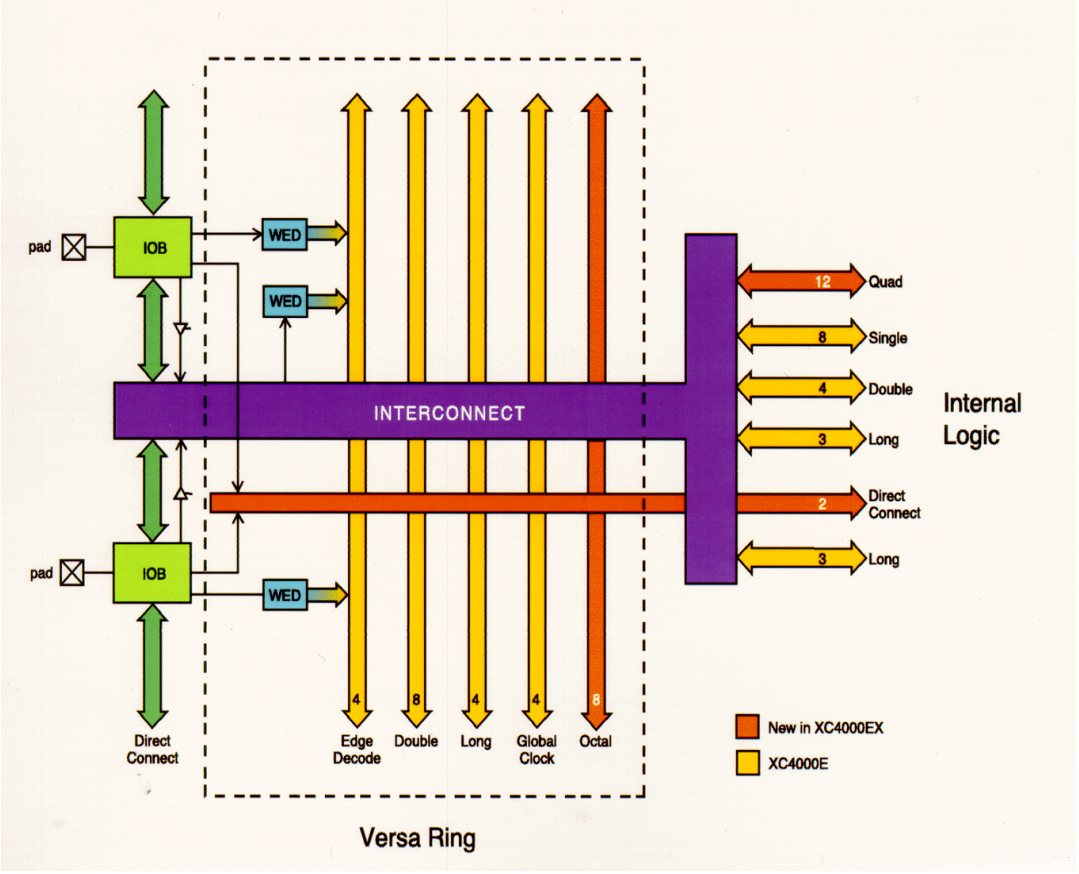

Figura 4.10

4.1.4 Spartan IIÔ.

Nella Figura 4.11 è mostrato lo schema della CLB della Spartan II della Xilinx. Questa FPGA è apparsa sul mercato nel Gennaio del 1998 e rappresenta una evoluzione delle XC4000; il numero di porte logiche di questa famiglia varia dalle 6.000 alle 200.000, la matrice di CLB varia da 8x12 a 28x42 e la dimensione del blocco della Ram varia dai 2Kbyte ai 7Kbyte. Questa famiglia di FPGA è quella che è stata adottata per lo sviluppo dell'intero progetto. La CLB della Spartan II, come si vede in figura, è costituita da due macro blocchi, detti slice, contenenti ognuno gli stessi elementi fondamentali della CLB della famiglia XC4000; una differenza fondamentale con la famiglia XC4000 è dovuta a una maggiore predisposizione per la realizzazione di circuiti aritmetici al suo interno: questo è dovuto ai segnali CIN e COUT (Carry In e Carry Out), che sono i bit di riporto di un eventuale sommatore.

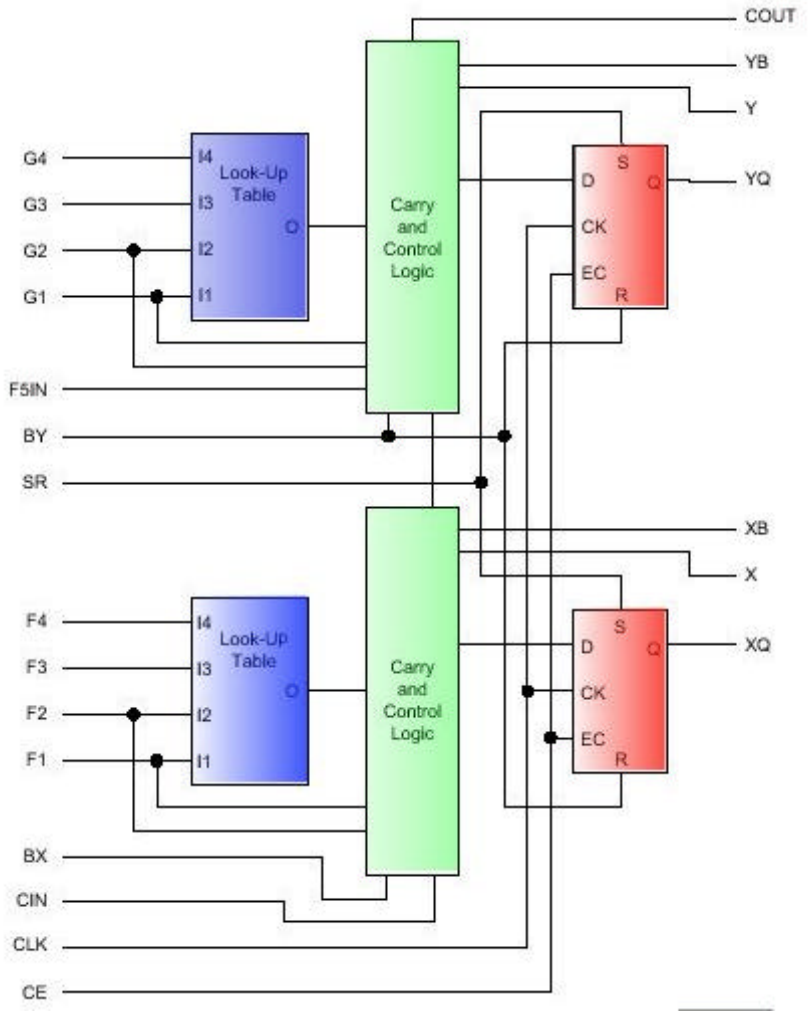

Figura 4.11

Un'altra differenza fondamentale è che le LUT della Spartan II possono implementare uno shift register a 16 bit quindi, essendoci due slice in ogni CLB contenenti ognuno due LUT, in ogni CLB possono essere configurati quattro shift register a 16 bit. Le Spartan II includono inoltre diversi blocchi di memoria RAM organizzate in colonne, come si può vedere dalla Figura 4. 12; tutti i modelli della Spartan II contengono due di queste colonne, una lungo ogni bordo verticale.

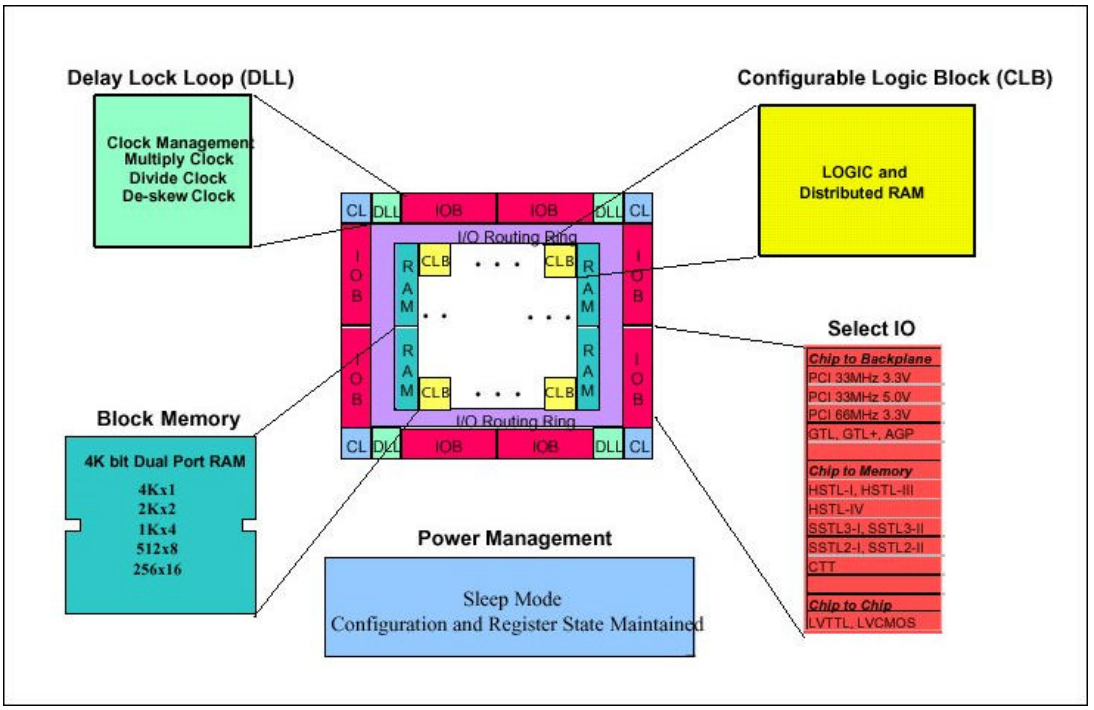

Figura 4. 12

Ogni blocco di memoria RAM, come si vede dalla Figura 4.13, è una Dual Port RAM da 4096 bit con dei segnali di controllo indipendenti per ogni uscita. L'ampiezza della parola trattata dalla RAM può essere configurata indipendentemente tramite una conversione dell'ampiezza del bus. Facendo un paragone con le attuali CPU, questa RAM può essere considerata come l'analogo della cache L2 (secondo livello), cioè una memoria molto veloce interna al microprocessore. Anche all'interno delle FPGA della famiglia XC4000 è possibile implementare della RAM configurando opportunamente le CLB ma, ovviamente, scegliendo una soluzione di questo tipo, queste CLB non sono più disponibili per scrivere altra logica.

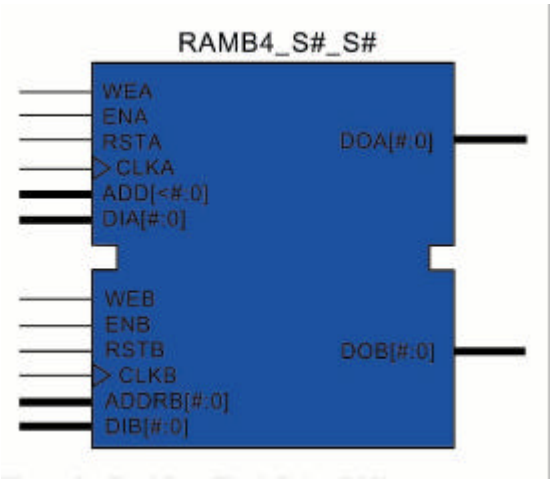
RAMB4_S#_S#
WEA
ENA
RSTA
CLKA
ADD[<#:0]
DIA[#:0]
DOA[#:0]
WEB
ENB
RSTB
CLKB
ADDRB[#:0]
DIB[#:0]
DOB[#:0]

Figura 4.13

Capitolo 5

Progetto del simulatore Monte Carlo Hardware

5.1 Introduzione.

Come discusso nei capitoli precedenti, è stato sviluppato un algoritmo di simulazione, analogo a quello descritto nel capitolo 3 e realizzato in C, tramite l'utilizzo di logiche programmabili quali le FPGA Spartan II della Xilinx; la programmazione di questi circuiti integrati viene realizzata caricando al loro interno un file binario che contenga tutte le informazioni necessarie per connettere correttamente tra loro gli oggetti contenuti al suo interno.

La creazione di questo file binario è stata possibile tramite l'utilizzo del software "Foundation 3.1i", distribuito sempre dalla Xilinx; Foundation permette di descrivere l'architettura del circuito elettronico che si desidera configurare all'interno della FPGA in diversi modi:

- Tramite degli schematici veri e propri nei quali sono rappresentati concretamente i singoli componenti, con i loro ingressi e le loro uscite, e i collegamenti tra loro;

- Tramite dei diagrammi di Moore nei quali vengono indicati i vari stati della macchina, le uscite di ogni blocco che la compone in quello stato e le condizioni che decidono la transizione da uno stato al successivo;
- Tramite la compilazione di listati nei quali la macchina può essere descritta sotto diversi aspetti: dal punto di vista comportamentale, combinatorio o sequenziale, con costrutti tipici di molti linguaggi di programmazione quali il C.

Di questi, vista la complessità del progetto che si è voluto realizzare, è stata scelta la descrizione tramite la compilazione di listati, la quale permette una maggiore elasticità nel fare delle variazioni durante la fase di realizzazione e consente di avere una visione più immediata del comportamento dei vari blocchi sotto diverse sollecitazioni.

Foundation permette la descrizione tramite listati in tre linguaggi: Abel, VHDL e Verilog; sebbene siano molto simili tra loro, si è preferito utilizzare il linguaggio Verilog in quanto esso è quello che permette una maggiore semplicità nella descrizione dei componenti.

Foundation dispone anche di un simulatore tramite il quale si è potuto verificare il corretto funzionamento della macchina una volta realizzata e si sono potute ottenere le misure del tempo di esecuzione, descritte nei paragrafi successivi.

5.2 Il generatore di numeri pseudo – casuali.

Abbiamo visto che il cuore di una macchina Monte Carlo è il generatore di numeri casuali; nei linguaggi di programmazione software, come il Basic e il C, l'estrazione continua di numeri casuali si realizza facilmente chiamando un'apposita funzione del linguaggio, come già detto nei capitoli precedenti.

Quando, però, si deve realizzare una macchina hardware di questo tipo, la generazione di numeri casuali diventa più complicata; usualmente non si ricorre all'utilizzo di "sorgenti" di eventi casuali ma si preferisce far uso di meccanismi deterministici attraverso i quali simulare la generazione di sequenze casuali: questo assicura la riproducibilità dell'esperimento virtuale, caratteristica indispensabile per il "debug" dei codici, oltre che un'elevata efficienza e stabilità dell'intero meccanismo.

Tra le innumerevoli tecniche proposte per la generazione di sequenze pseudo-casuali via software, quali l'immagazinamento in memoria di decine di migliaia numeri tratti a caso da rapporti su censimenti, poche si adattano bene ad un'implementazione direttamente realizzata su componenti hardware.

5.2.1 Generatore a registri di scorrimento retroazionato.

Un metodo per generare numeri casuali che consenta una realizzazione efficiente e veloce direttamente sull'hardware è basato sui registri a scorrimento (shift register) ed è noto come "Generatore a registri di scorrimento retroazionato" (Feedback shift register generator); esso è costituito da un registro a scorrimento sincrono a n stadi retroazionato attraverso una opportuna funzione logica a m ingressi, che nel nostro caso è rappresentata da una porta EX-OR, come si vede nella Figura 5.1.

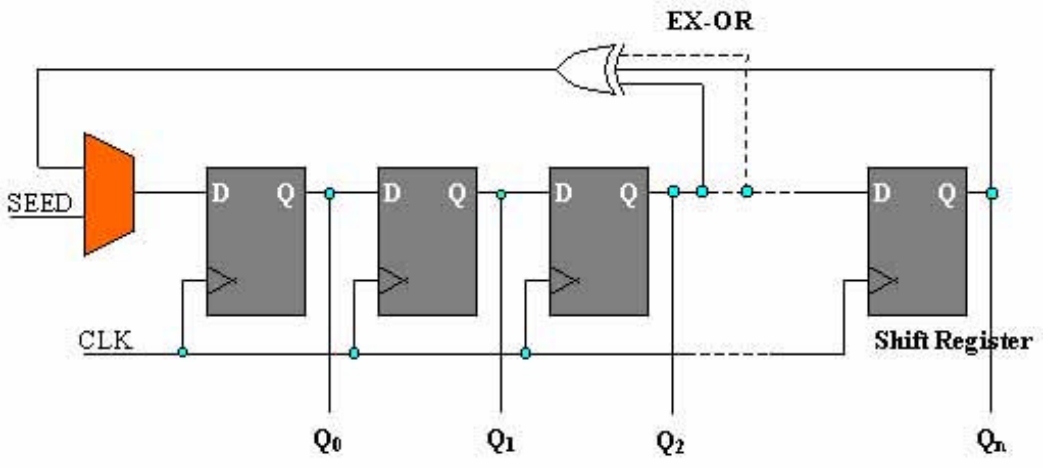

Figura 5.1

La sequenza di numeri generata da questa rete combinatoria dipende unicamente dalla parola inizialmente memorizzata nei registri in fase di inizializzazione, detta "seme della generazione" (seed); se tutti gli n flip-flop del registro a scorrimento, rappresentanti ognuno una singola cella di memoria contenente un bit, contenessero il numero zero, allora anche il segnale seriale d'ingresso del registro, uscita dell'EX-OR, sarebbe zero e pertanto l'uscita parallela del registro (Q_n, Q_{n-1}, …, Q_1, Q_0), da dove viene prelevato il numero pseudo-casuale generato, rimarrebbe zero per tutta la durata della simulazione: è dunque necessario impostare nel registro una configurazione di bit iniziale diversa da zero.

Fissato il seme, la porta EX-OR genera l'ingresso seriale combinando un gruppo di m bit tra le n cifre in uscita dai vari flip-flop e ad ogni ciclo di clock la cifra scorre di un passo, perdendo il bit più significativo e acquisendo il nuovo bit in uscita dall'EX-OR. La successione di parole a n bit prodotta in questo modo all'uscita parallela dei registri costituisce la sequenza pseudo-casuale prodotta dal generatore.

Il gruppo di m bit utilizzati per la generazione dell'ingresso seriale deve essere scelto opportunamente; da esso dipende il periodo della sequenza prodotta dal generatore: infatti, dopo un certo numero di cicli di clock capiterà che la parola in uscita dal registro sia identica al seme che ha generato la sequenza iniziale e, quindi, da questo punto in poi, la sequenza si ripeterà identicamente per tutta la durata della simulazione.

Fissata la dimensione della parola in uscita dal generatore, è possibile determinare quali bit devono essere messi in EX-OR per ottenere la sequenza di periodo massimo; il massimo periodo ottenibile è dato ovviamente da $2^n - 1$, dove 2^n è il numero massimo di combinazioni che si possono ottenere da una parola di n bit da cui si deve sottrarre quella con tutti i bit pari a zero. Nella Tabella 5.1 sono riportati i bit di cui fare l'EX-OR in funzione della dimensione della parola del registro a scorrimento per ottenere la sequenza con il periodo massimo.

n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

Tabella 5.1

5.2.2 Generatore di McLaren – Marsaglia.

Nella macchina che abbiamo realizzato è stato utilizzato un generatore di numeri pseudo-casuali che sfrutta l'idea del generatore a scorrimento retroazionato, migliorandone la bontà della sequenza, ottenuto utilizzando un meccanismo che fu proposto da McLaren e Marsaglia nel 1965. Il meccanismo consiste nell'utilizzo di un primo

generatore a scorrimento retroazionato, per la produzione della sequenza di numeri pseudo-casuali, e di un secondo generatore che mescoli la sequenza prodotta dal primo in maniera casuale; in questo modo, i numeri casuali prodotti non sono forniti nell'ordine in cui sono stati generati, a con un ordine diverso, determinato anch'esso casualmente. Come dimostrato dagli autori, la sequenza casuale così ottenuta risulta avere caratteristiche migliori della sequenza di partenza; si può quindi dire, in un certo senso, che il meccanismo descritto "aumenta la casualità" dei numeri prodotti.

Nella Figura 5.2 è mostrato lo schema a blocchi della realizzazione del generatore di numeri casuali di McLaren – Marsaglia. Come si può vedere, mentre il primo generatore scrive i numeri prodotti nelle celle di memoria di una RAM, il secondo utilizza i numeri da lui prodotti per leggere casualmente dalla memoria; questo processo richiede quindi una fase di inizializzazione in i numeri prodotti dal primo generatore sono scritti sequenzialmente nella memoria. Se i due generatori hanno rispettivamente dimensioni n e m, si può dimostrare che il massimo periodo ottenibile è dato da $2^{n+m} - 1$.

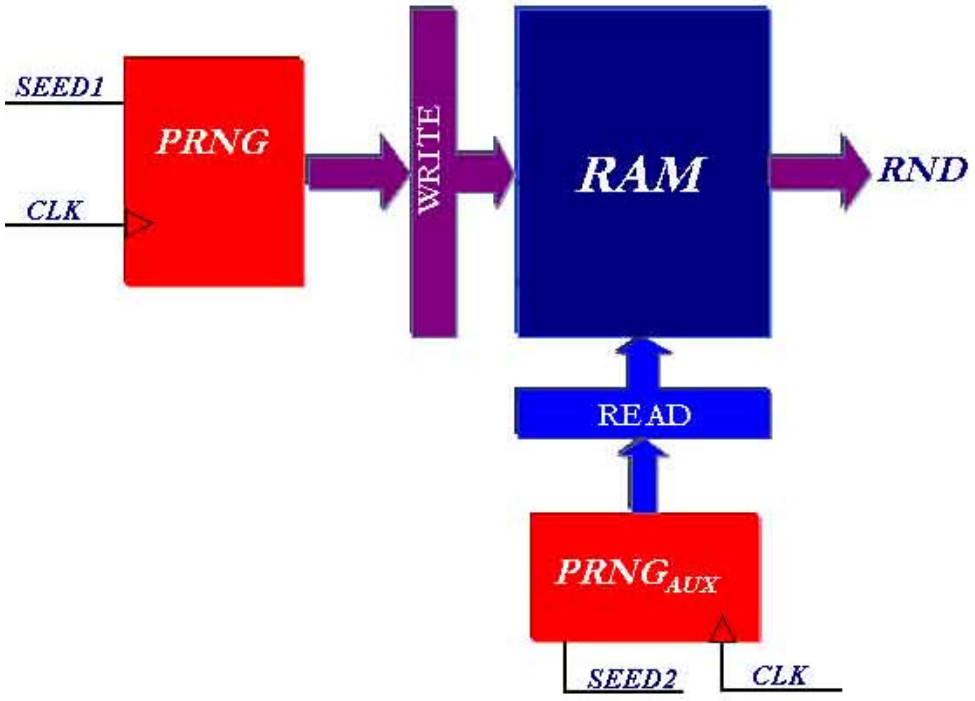

Figura 5.2

5.3 Calcolo dei Seni e Coseni degli angoli di scattering.

Un altro punto cruciale dello sviluppo dell'algoritmo di simulazione software in hardware, oltre la generazione di numeri pseudo-casuali, è stato il calcolo dei Seni e Coseni degli angoli di scattering Θ e Φ; infatti, il modo in cui generalmente si procede in questi casi è la costruzione di una Look Up Table (LUT), cioè una memoria in cui sono conservati i valori, precedentemente calcolati, di queste grandezze; questi valori non possono ovviamente essere continui, ma sono calcolati a step costanti, scelti in base alla precisione offerta dalla dimensione della parola trattata dalla macchina.

Riempita la LUT, ogni volta che sul bus degli indirizzi della RAM, dove sono memorizzati i valori dei Seni e Coseni, viene inviato il valore dell'angolo, questa manda in uscita i corrispettivi valori del Seno e del Coseno.

Si capisce subito , però, che se da una parte questa tecnica è molto veloce (in quanto richiede soltanto il tempo di accesso alla lettura della RAM) dall'altra richiede un notevole spazio di memoria; facendo un rapido calcolo: se la dimensione della parola che rappresenta il Seno e il Coseno è di 16 bit, allora lo spazio in memoria necessario per conservare tutti i valori di queste grandezze è pari a:

$$\left[(2^{16})\cdot(2^{4})\cdot(2^{2})\right]\ \text{bit} = 2^{22}\,\text{bit} = 2^{19}\,\text{byte} \cong 1\ \ \text{Mbyte} \qquad (5.1)$$

dove 2^{16} sono i possibili valori di una cifra binaria a 16 bit, 2^4 è la dimensione della parola da memorizzare, e 2^2 sono le grandezze in questione (Senθ, Cosθ, Senφ e Cosφ).

Si è preferito allora ricorrere ad un algoritmo iterativo molto usato in applicazioni che riguardano i processori dei segnali radar e la robotica: l'algoritmo CORDIC.

5.3.1 L'algoritmo CORDIC.

Il calcolatore CORDIC (COordinate Rotation DIgital Computer) è una macchina digitale dedicata a particolari calcoli effettuati in tempo reale. In questa macchina viene utilizzata una tecnica di calcolo che è particolarmente adatta per risolvere le relazioni trigonometriche che sono implicate nella rotazione del piano di coordinate e nella conversione da coordinate cartesiane a coordinate polari. CORDIC è un computer vero e proprio: esso contiene una speciale unità aritmetica seriale consistente di tre registri a scorrimento, tre sommatori – sottrattori, e delle interconnessioni speciali. Tramite l'utilizzo di una prescritta sequenza di eventuali somme o sottrazioni, l'unità aritmetica di CORDIC può essere controllata in maniera da risolvere sia il seguente sistema di equazioni:

$$\begin{cases} Y' = K\left(Y \cdot \mathrm{Cos}\ l + X \cdot \mathrm{Sen}\ l\right) \\ X' = K\left(X \cdot \mathrm{Cos}\ l - Y \cdot \mathrm{Sen}\ l\right) \end{cases} \qquad (5.2)$$

che:

$$\begin{cases} R = K\sqrt{X^2 + Y^2} \\ q = \mathrm{Tg}^{-1}\left(\dfrac{Y}{X}\right) \end{cases} \qquad (5.3)$$

dove K è una certa costante. Questa speciale unità aritmetica è anche adatta ad eseguire altri calcoli quali moltiplicazioni, divisioni e conversioni da numeri binari ad altri codici; tuttavia noi discuteremo soltanto gli algoritmi trigonometrici utilizzati in questo computer e la loro realizzazione circuitale.

La tecnica di calcolo CORDIC fu sviluppata soprattutto per essere applicata a un computer digitale che doveva risolvere dei calcoli in tempo reale, nel quale la maggior parte dei calcoli richiedeva una soluzione programmata delle relazioni trigonometriche riguardanti le equazioni di navigazione e un'alta frequenza nella soluzione delle relazioni di trasformazione di coordinate. Un prototipo di questa macchina, CORDIC I, è stato progettato e costruito a Convair, Fort Worth. Sebbene CORDIC I possa essere classificato come una unità logica – aritmetica vera e propria, la sua struttura non si basa su quella dei microprocessori ad uso generale.

5.3.2 Descrizione funzionale.

Per spiegare come il computer CORDIC risolva le operazioni trigonometriche che gli vengono richieste dobbiamo fare una distinzione tra due differenti metodi di calcolo: la rotazione e la vettorializzazione. Nel metodo della rotazione, date le componenti di un vettore e un angolo di rotazione, vengono calcolate le componenti del vettore risultante dopo una rotazione pari all'angolo dato. Nel metodo della vettorializzazione invece, date le componenti di un vettore, viene calcolato il suo modulo e l'angolo che esso forma con l'asse X di riferimento. Analogamente a come quando uno risolve manualmente questo problema, il dispositivo di rotazione viene utilizzato nel metodo della vettorializzazione: il vettore di origine è infatti ruotato finché l'angolo con l'asse X di riferimento non vale zero, in modo che il valore della componente X del vettore coincida col modulo del vettore originale.

In pratica, la tecnica di base usata da CORDIC sia nel metodo della rotazione che in quello della vettorializzazione è una sequenza passo dopo passo di pseudo rotazioni che risultano alla fine in una rotazione completa dell'angolo dato (metodo della rotazione) o in un angolo del vettore con l'asse X di riferimento nullo (metodo della vettorializzazione).

È necessario che l'incremento angolare della rotazione sia calcolato in un ordine decrescente. Ci sono diversi valori permessi che possono essere scelti per la grandezza angolare del primo step di rotazione: la grandezza dell'angolo che generalmente viene scelto per la prima rotazione è 90°; il sistema da risolvere per ottenere le coordinate componenti Y_1 e X_1 risultanti dopo una rotazione di ± 90° è semplicemente:

$$\begin{cases} Y_2 = \pm X_1 = R_1 \mathrm{Sen}\,(q_1 \pm 90°) \\ X_2 = \mp Y_1 = R_1 \mathrm{Cos}\,(q_1 \pm 90°) \end{cases} \qquad (5.4)$$

I successivi step di calcolo possono essere chiariti esaminando le relazioni, mostrate in figura, che sono implicate in un tipico step di rotazione. Consideriamo due date coordinate componenti X_i e Y_i nel piano del sistema di coordinate rappresentato in figura, dove l'indice i rappresenta il numero dello step. Le componenti X_i e Y_i sono

associate all'i-esimo step e descrivono un vettore di modulo R_i posto con un certo angolo θ_i rispetto all'asse X di riferimento, secondo la relazione:

$$\begin{cases} Y_i = R_i \cdot \mathrm{Sen}\theta_i \\ X_i = R_i \cdot \mathrm{Cos}\theta_i \end{cases} \qquad (5.5)$$

Nella Figura 5.3, l'angolo α_i è il particolare angolo di rotazione associato a ogni step del calcolo, la cui espressione generale è:

$$\alpha_i = \mathrm{Tg}^{-1}\left[2^{-(i-2)}\right] \qquad (5.6)$$

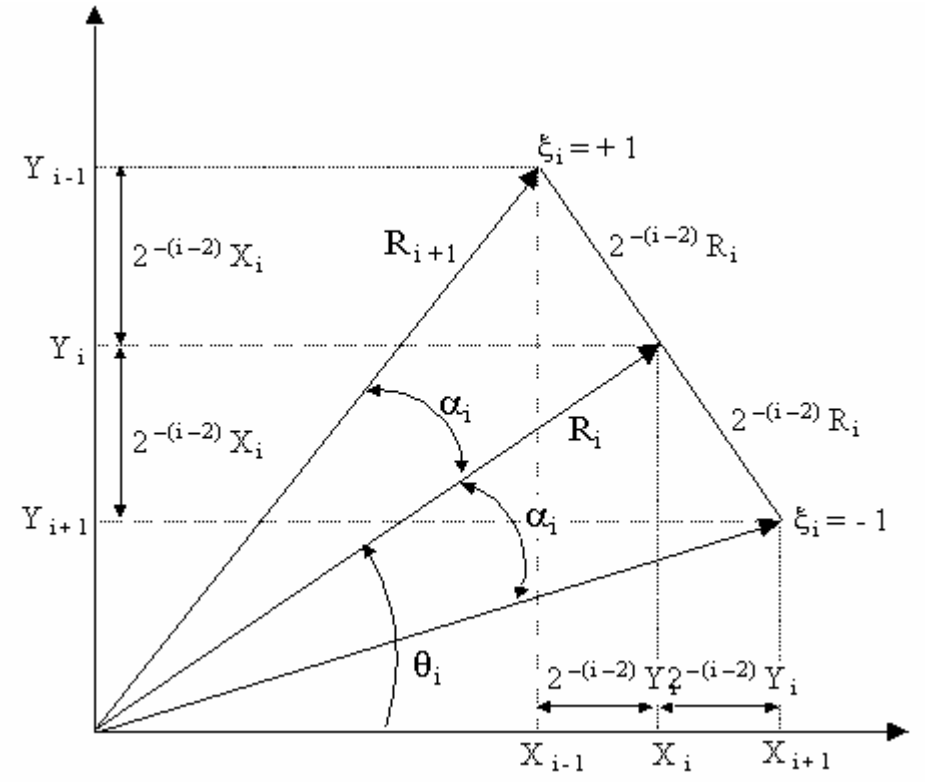

Figura 5.3

La grandezza caratteristica di ogni α_i è tale che una rotazione di coordinate componenti di un angolo ($\pm \alpha_i$) possa essere calcolata semplicemente con uno shift e una somma. Le due scelte di una rotazione positiva o negativa sono mostrate in figura; l'espressione generale per le componenti ruotate è:

$$\begin{cases} Y_{i+1} = \sqrt{1+2^{-2(i-2)}} \cdot R_i \cdot \text{Sen}\,(q_i \pm a_i) = Y_i \pm \left[2^{-(i-2)} \cdot X_i\right] \\ X_{i+1} = \sqrt{1+2^{-2(i-2)}} \cdot R_i \cdot \text{Cos}\,(q_i \pm a_i) = X_i \mp \left[2^{-(i-2)} \cdot Y_i\right] \end{cases} \quad (5.7)$$

da cui si vede che il modulo del vettore diventa, per effetto di questa trasformazione:

$$R_{i+1} = \left[\sqrt{1+2^{-2(i-2)}}\right] \cdot R_i \quad (5.8)$$

Si può osservare che, restringendo i possibili valori di α_i a quelli permessi dalla sua espressione $a_i = \text{Tg}^{-1}\left[2^{-(i-2)}\right]$, le espressioni di X_i e Y_i possono essere calcolate facendo due simultanee operazioni di shift e somma; questa è la relazione fondamentale sulla quale è fondata questa tecnica di calcolo computazionale. L'operazione di sommare (o sottrarre) un valore shiftato di X_i a Y_i per ottenere Y_{i+1} mentre simultaneamente si sottrae (o si somma) un valore shiftato di Y_i a X_i per ottenere X_{i+1} è chiamata " somma incrociata".

Mentre le espressioni di Y_{i+1} e X_{i+1} non sono delle rotazioni perfette poiché incrementano il modulo del vettore di una quantità pari a $\sqrt{1+2^{-2(i-2)}}$, entrambe le due scelte nella direzione di rotazione producono lo stesso incremento nel modulo. Se perciò, a ogni passo, le coordinate sono sempre ruotate di un angolo α_i positivo o negativo, allora l'incremento nel modulo può essere considerato una costante. Questa assunzione preclude la scelta di rotazioni nulle ad ogni step. Per identificare la scelta fatta in un particolare step, la notazione $\pm$ può essere rappresentata dall'operatore binario ξ_i il quale può assumere solo i valori +1 o –1. Questa sostituzione produce l'espressione generale:

$$\begin{cases} Y_{i+1} = \sqrt{1+2^{-2(i-2)}} \cdot R_i \cdot \text{Sen}\,(q_i + x_i a_i) = Y_i + \left[x_i \cdot 2^{-(i-2)} \cdot X_i\right] \\ X_{i+1} = \sqrt{1+2^{-2(i-2)}} \cdot R_i \cdot \text{Cos}\,(q_i + x_i a_i) = X_i + \left[x_i \cdot 2^{-(i-2)} \cdot Y_i\right] \end{cases}$$

(5.9)

Analogamente, dopo aver completato lo step di rotazione nel quale si sono ottenuti i termini (i + 1)–esimi, i termini (i + 2)–esimi possono essere ricavati dai termini precedenti tramite la relazione:

$$\begin{cases} Y_{i+2} = \sqrt{1+2^{-2(i-1)}}\sqrt{1+2^{-2(i-2)}} \cdot R_i \cdot \mathrm{Sen}\,(q_i + x_i a_i + x_{i+1} a_{i+1}) \\ X_{i+2} = \sqrt{1+2^{-2(i-1)}}\sqrt{1+2^{-2(i-2)}} \cdot R_i \cdot \mathrm{Cos}\,(q_i + x_i a_i + x_{i+1} a_{i+1}) \end{cases}$$

(5. 10)

Similmente, gli step di pseudo rotazione possono essere ripetuti per un qualunque numero finito di volte prestabilito, senza alcuna preoccupazione nei riguardi del valore assegnato a ξ.

Consideriamo le coordinate componenti iniziali X_1 e Y_1 dove:

$$\begin{cases} Y_1 = R_1 \cdot \mathrm{Sen}\ q_1 \\ X_1 = R_1 \cdot \mathrm{Cos}\ q_1 \end{cases} \qquad \textbf{(5. 11)}$$

Avendo stabilito il primo e più significativo step come una rotazione di $\pm 90°$ e avendo prefissato il numero di step a *n*, l'espressione finale delle coordinate sarà:

$$\begin{cases} Y_{n+1} = \left[\sqrt{1+2^{-0}}\sqrt{1+2^{-2}} \ldots \sqrt{1+2^{-2(n-2)}}\right] \cdot R_1 \cdot \mathrm{Sen}(q_1 + x_1 a_1 + \ldots + x_n a_n) \\ X_{n+1} = \left[\sqrt{1+2^{-0}}\sqrt{1+2^{-2}} \ldots \sqrt{1+2^{-2(n-2)}}\right] \cdot R_1 \cdot \mathrm{Cos}(q_1 + x_1 a_1 + \ldots + x_n a_n) \end{cases}$$

5. 12

L'aumento di modulo delle componenti per un particolare valore di *n* è una costante che rappresenteremo con una lettera K. Il valore scelto per *n* è una funzione della precisione di calcolo desiderata e può essere una costante che varia da computer a computer; ad esempio, se *n* = 24 allora K = 1,646760255.

I necessari componenti funzionali e il flusso di informazioni per allestire la somma incrociata sono associati ai registri X e Y mostrati in Figura 5.4.

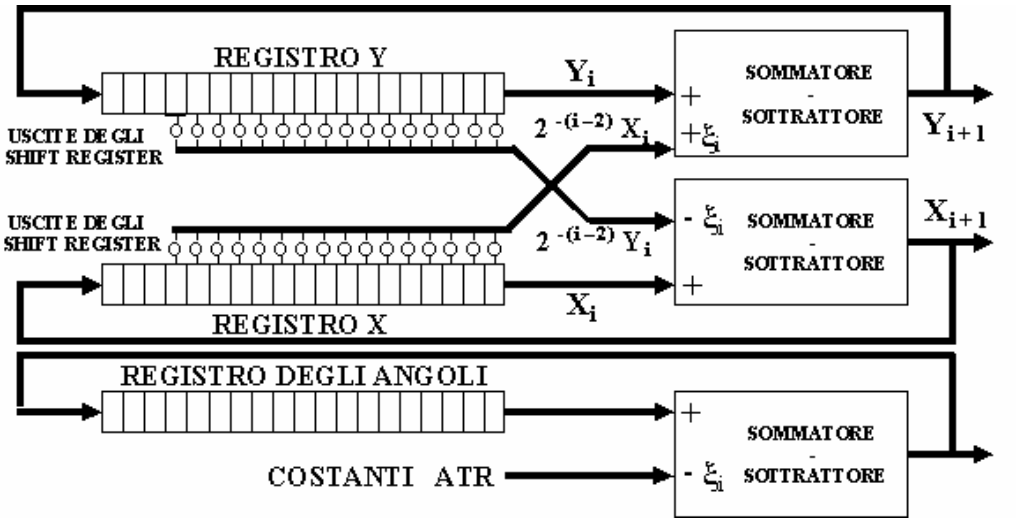

Figura 5.4

Non abbiamo ancora detto come la sequenza che abbiamo visto di step di pseudo rotazioni possa essere controllata per effettuare la desiderata rotazione completa. Osservando le espressioni di X_{n+1} e Y_{n+1} si può vedere che, per una rotazione di un sistema di coordinate componenti X_1 e Y_1 di un dato angolo (come richiesto nel metodo della rotazione), è necessario ottenere le espressioni:

$$\begin{cases} Y_{n+1} = K \cdot R_1 \cdot \mathrm{Sen}(q_1 + l) \\ X_{n+1} = K \cdot R_1 \cdot \mathrm{Cos}(q_1 + l) \end{cases} \quad (5.13)$$

dove:

$$l = x_1 a_1 + x_2 a_2 + \ldots + x_n a_n \quad (5.14)$$

Per come abbiamo definito il metodo della vettorializzazione dovrà essere:

$$-q_1 = x_1 a_1 + x_2 a_2 + \ldots + x_n a_n \quad (5.15)$$

Le successioni che troviamo nelle espressioni di λ e θ_1 formano una speciale rappresentazione equivalente all'angolo desiderato; in questa successione i valori dei primi termini sono pari a:

$$\begin{cases} a_1 = 90° \\ a_2 = \mathrm{Tg}^{-1}\left(2^{-0}\right) = 45° \\ a_3 = \mathrm{Tg}^{-1}\left(2^{-1}\right) \approx 26.5° \\ \dots \\ a_i = \mathrm{Tg}^{-1}\left[2^{-(i-2)}\right] \end{cases} \qquad (5.16)$$

I termini α_i sono detti "costanti ATR" (Arc Tangent Radix) e sono delle costanti precalcolate e memorizzate nel computer; i termini ξ_i sono invece chiamati "cifre ATR" e sono calcolati durante ogni step.

Nel computer CORDIC le cifre ATR vengono determinate sequenzialmente, iniziando dalla più significativa, e sono usate per controllare l'eventuale azione dei sommatori – sottrattori nell'unità aritmetica; andiamo a vedere come la rappresentazione in codice binario della cifre ATR, $\xi_1\xi_2\xi_3\ldots\xi_n$, può essere determinata per ogni dato valore dell'angolo θ o λ . Per prima cosa, per ogni valore di θ o λ, ci deve essere almeno un insieme di valori degli operatori ξ che soddisfa l'espressione di θ o λ; Secondo, deve esistere una semplice tecnica per determinare le cifre ATR in codice binario che soddisfino queste equazioni. Le seguenti relazioni sono necessarie e sufficienti affinché ogni sequenza di costanti α_i soddisfi le condizioni di sopra:

$$\begin{cases} \begin{cases} |l| \le a_1 + a_2 + a_3 + \ldots + a_n + a_n \\ \text{oppure} \\ |q| \le a_1 + a_2 + a_3 + \ldots + a_n + a_n \end{cases} \\ a_i \le a_{i+1} + a_{i+2} \ldots + a_n + a_n \end{cases} \qquad (5.17)$$

Affinché vengano soddisfatte le condizioni a contorno:

$$\begin{cases} x = +1 \text{ o } -1 \\ a_i = 90° \end{cases} \qquad (5.18)$$

è necessario che θ o λ possano assumere soltanto valori compresi tra –180° e +180°. Questa costrizione non comporta nessuna speciale considerazione se viene utilizzata la rappresentazione in complemento a due.

Tramite l'utilizzo di un sommatore – sottrattore e un registro, che in figura è indicato come "registro degli angoli", la relazione $Y_{n+1} = K R_1 Sen(\theta_1 + \lambda)$, che abbiamo visto descrivendo il metodo della rotazione, può essere soddisfatta eseguendo queste semplici istruzioni:

- annotando il segno dell'angolo di rotazione (o del rimanente se i > 1);
- sottraendo o sommando all'angolo la costante ATR corrispondente a quel particolare step.

A ogni passo la relazione che viene applicata è:

$$|l_{i+1}| = ||l_i| - a_i|$$

L'esecuzione del primo step alla prima delle due condizioni necessarie e sufficienti viste prima risulta in:

$$-a_i \le |l| - a_i \le a_2 + a_3 + \ldots + a_n + a_n \qquad (5.19)$$

e:

$$|l_2| \equiv ||l_1| - a_i| \le a_2 + a_3 + \ldots + a_n + a_n \qquad (5.20)$$

Continuando con questo processo si trova che:

$$|l_{n+1}| \le a_n \qquad (5.21)$$

Questa equazione può essere usata per dimostrare che ciò che rimane nel registro degli angoli tende a zero nel metodo delle rotazioni.

La sequenza dei segni delle operazioni usati per far tendere λ a zero è l'inverso dell'equivalente codice di cifre ATR per l'angolo originale λ_1. Più semplicemente, il codice di cifre ATR di ogni step è uguale al segno della quantità nel registro degli angoli prima di ogni step. Perciò, simultaneamente a ogni step in cui si fa tendere λ a zero nel registro degli angoli, il codice di cifre ATR può essere usato per controllare

lo step delle somme incrociate nei registri X e Y al fine di effettuare una rotazione delle coordinate componenti di un uguale incremento angolare.

La prova della convergenza dell'angolo θ_{n+1} a zero, che è necessaria nel metodo della vettorializzazione, può essere ottenuta sostituendo λ con θ nella equazione:

$$|l_{i+1}| = \|l_i| - a_i| \qquad (5.22)$$

tramite la relazione:

$$|l_{n+1}| \leq a_n \qquad (5.23)$$

Nel metodo della vettorializzazione, il segno dell'angolo θ_i si ottiene misurando il segno di Y_i. La sequenza dei segni di Y_i è l'inverso del codice ATR per l'effettiva rotazione effettuata sulle componenti X_1 e X_1. Durante ogni operazione di somma incrociata nei registri X e Y, la corrispondente costante ATR può essere eventualmente sommata o sottratta, a seconda del valore di ξ_i, a una somma accumulata nel registro degli angoli in modo tale che, alla fine della sequenza di calcolo, quando $\theta_{n+1} = 0$ la quantità nel registro degli angoli sia uguale all'angolo θ_1 di partenza delle coordinate componenti Y_1 e X_1.

I risultati a ogni step di una tipica sequenza di calcolo per una rotazione sono mostrati nella Tabella 5.2. La notazione in complemento a due è utilizzata per tutte le quantità e, per semplicità, le quantità shiftate sono semplicemente troncate senza essere arrotondate. I risultati passo dopo passo di una tipica sequenza di calcolo per il metodo della vettorializzazione sono invece mostrate nella Tabella 5.3.

La tecnica di calcolo CORDIC quindi è particolarmente adatta a essere usata in macchine dedicate a calcoli specifici, dove la maggior parte delle operazioni richiedono delle relazioni trigonometriche; le operazioni di rotazione e di vettorializzazione possono essere considerate delle routine di lunghezza costante in cui il numero di parole elaborate alla volta è uguale alla lunghezza della parola stessa.

Quello che si pensa è che algoritmi simili a questo, basati su questo fondamentale concetto di calcolo, possano essere sviluppati per molte altre applicazioni.

TABELLA I: TIPICA SEQUENZA DI CALCOLO PER IL METODO DELLA ROTAZIONE		
REGISTRO Y	REGISTRO X	**REGISTRO DEGLI ANGOLI**
$Y_1 = 0.0101110$	$X_1 = 1.1000101$	$\lambda = 0.1100101$
$+ \underline{1.1000101}$	$- \underline{0.0101110}$	$- \underline{0.1000000}$ $(Tg^{-1} \infty)$
$= 1.1000101$	$= 1.1010010$	$= 0.0100101$
$+ \underline{1.1010010}$	$- \underline{1.1000101}$	$- \underline{0.0100000}$ $(Tg^{-1} 1)$
$= 1.0010111$	$= 0.0001101$	$= 0.0000101$
$+ \underline{0.0000110}$	$- \underline{1.1001011}$	$- \underline{0.0010010}$ $(Tg^{-1} 2^{-1})$
$= 1.0011101$	$= 0.1000010$	$= 1.1110011$
$- \underline{0.0010000}$	$+ \underline{1.1100111}$	$+ \underline{0.0001001}$ $(Tg^{-1} 2^{-2})$
$= 1.0001101$	$= 0.0101001$	$= 1.1111100$
$- \underline{0.0000101}$	$+ \underline{1.1110001}$	$+ \underline{0.0000101}$ $(Tg^{-1} 2^{-3})$
$= 1.0001000$	$= 0.0011010$	$= 0.0000001$
$+ \underline{0.0000001}$	$- \underline{1.1111000}$	$- \underline{0.0000010}$ $(Tg^{-1} 2^{-4})$
$= 1.0001001$	$= 0.0100010$	$= 1.1111111$
$- \underline{0.0000001}$	$+ \underline{1.1111100}$	$+ \underline{0.0000001}$ $(Tg^{-1} 2^{-5})$
$= 1.0001000$	$= 0.0011110$	$= 0.0000000$

Tabella 5.2

TABELLA II: TIPICA SEQUENZA DI CALCOLO PER IL METODO DELLA VETTORIALIZZAZIONE		
REGISTRO Y	REGISTRO X	**REGISTRO DEGLI ANGOLI**
$Y_1 = 0.0101110$	$X_1 = 1.1000101$	0.0000000
$- \underline{1.1000101}$	$+ \underline{0.0101110}$	$+ \underline{0.1000000}$ $(Tg^{-1} \infty)$
$= 0.0111011$	$= 0.0101110$	$= 0.1000000$
$- \underline{0.0101110}$	$+ \underline{0.0111011}$	$+ \underline{0.0100000}$ $(Tg^{-1} 1)$
$= 0.0001101$	$= 0.1101001$	$= 0.1100000$
$- \underline{0.0110100}$	$+ \underline{0.0000110}$	$+ \underline{0.0010010}$ $(Tg^{-1} 2^{-1})$
$= 1.1011001$	$= 0.1101111$	$= 0.1110010$
$+ \underline{0.0011011}$	$- \underline{1.1110110}$	$- \underline{0.0001001}$ $(Tg^{-1} 2^{-2})$
$= 1.1110100$	$= 0.1111001$	$= 0.1101001$
$+ \underline{0.0001111}$	$- \underline{1.1111110}$	$- \underline{0.0000101}$ $(Tg^{-1} 2^{-3})$
$= 0.0000011$	$= 0.1111011$	$= 0.1100100$
$- \underline{0.0000111}$	$+ \underline{0.0000000}$	$+ \underline{0.0000010}$ $(Tg^{-1} 2^{-4})$
$= 1.1111100$	$= 0.1111011$	$= 0.1100110$
$+ \underline{0.0000011}$	$- \underline{1.1111111}$	$- \underline{0.0000001}$ $(Tg^{-1} 2^{-5})$
$= 1.1111111$	$= 0.1111100 = KR_1$	$= 0.1100101 = \theta$

Tabella 5.3

5.4 Schema ad alto livello della rete combinatoria.

Nella Figura 5.5 è mostrato lo schema a blocchi della rete combinatoria realizzata tramite la descrizione nel linguaggio Verilog. La dimensione scelta per le parole elaborate dai vari blocchi è stata di 16 bit: è nostra opinione che questa scelta consenta di avere una precisione sufficiente per la descrizione di un tipico problema di calcolo della dose in radioterapia, tenendo conto di quali sono le dimensioni lineari tipiche su cui si sviluppa la regione da trattare (circa 30 cm) e la dimensione della discretizzazione adottata per il calcolo della dose assorbita ($1mm^3$). Uno studio più accurato della dipendenza tra l'errore e il passo di discretizzazione (cioè il numero di bit adottati per rappresentare le diverse grandezze) è comunque necessario se si vuole più precisamente comprendere le potenzialità e le caratteristiche dell'approccio.

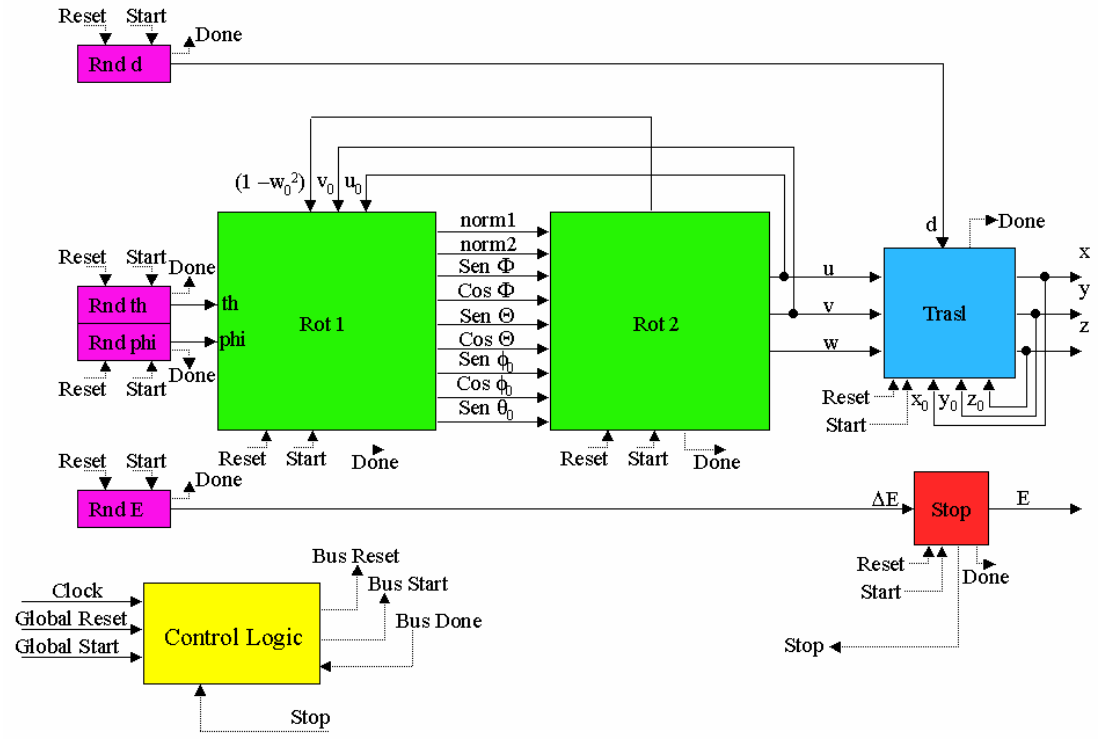

Figura 5.5

In questo schema si riconoscono quattro generatori di numeri pseudo-casuali (in viola), realizzati come descritto nei paragrafi precedenti, utilizzati per l'estrazione pseudo-casuale delle grandezze d (generata da Rnd d – distanza percorsa dal fotone

prima di interagire col mezzo), E (generata da Rnd E – energia persa dal fotone a causa dell'interazione), Θ e Φ (generati da Rnd th e Rnd phi – angoli di scattering).

L'intera rete combinatoria è pilotata da una logica sequenziale (in giallo) che gestisce il passaggio della macchina da uno stato al successivo in base a dei segnali di controllo detti "done" che riceve dai singoli blocchi componenti e a dei segnali di "start" e "reset" che invia sempre ai singoli blocchi.

Gli angoli Θ e Φ vengono processati dal primo blocco di rotazione (Rot1 – in verde) che ne calcola il Seno e il Coseno tramite l'algoritmo CORDIC precedentemente descritto; assieme ai valori dei Seni e Coseni di Θ e Φ, questo blocco restituisce anche i valori dei Seni e Coseni di θ_0 e φ_0 dove, al primo step, θ_0 e φ_0 hanno dei valori prefissati tali che la direzione iniziale del fotone sia lungo l'asse X.

Queste grandezze sono quindi elaborate dal secondo blocco di rotazione (Rot2 – in verde) dove sono calcolati i nuovi valori dei coseni direttori {u, v, w} a partire dai precedenti $\{u_0, v_0, w_0\}$.

L'ultimo blocco (Trasl – in blu) effettua infine la traslazione delle coordinate del fotone nella direzione determinata da {u, v, w} e restituisce in uscita le nuove coordinate che saranno immagazzinate nella RAM.

Ogni volta che sono state calcolate le nuove posizioni del fotone, il blocco di Stop (in rosso) calcola di quanto è diminuita l'energia di questo; l'energia persa a causa dello scattering viene estratta casualmente dal blocco Rnd E e passata al blocco di Stop, il quale fornisce in uscita il nuovo valore dell'energia dopo l'urto e, se questa è minore di un certo valore di soglia, invia all'unità di controllo dell'intera macchina un segnale che indica la fine della storia del fotone in questione.

5.5 Descrizione della macchina a stati.

Come detto, l'intera rete combinatoria appena descritta è pilotata da una logica di controllo sequenziale. Lo schema di questa macchina a stati è descritto nella Figura 5.6.

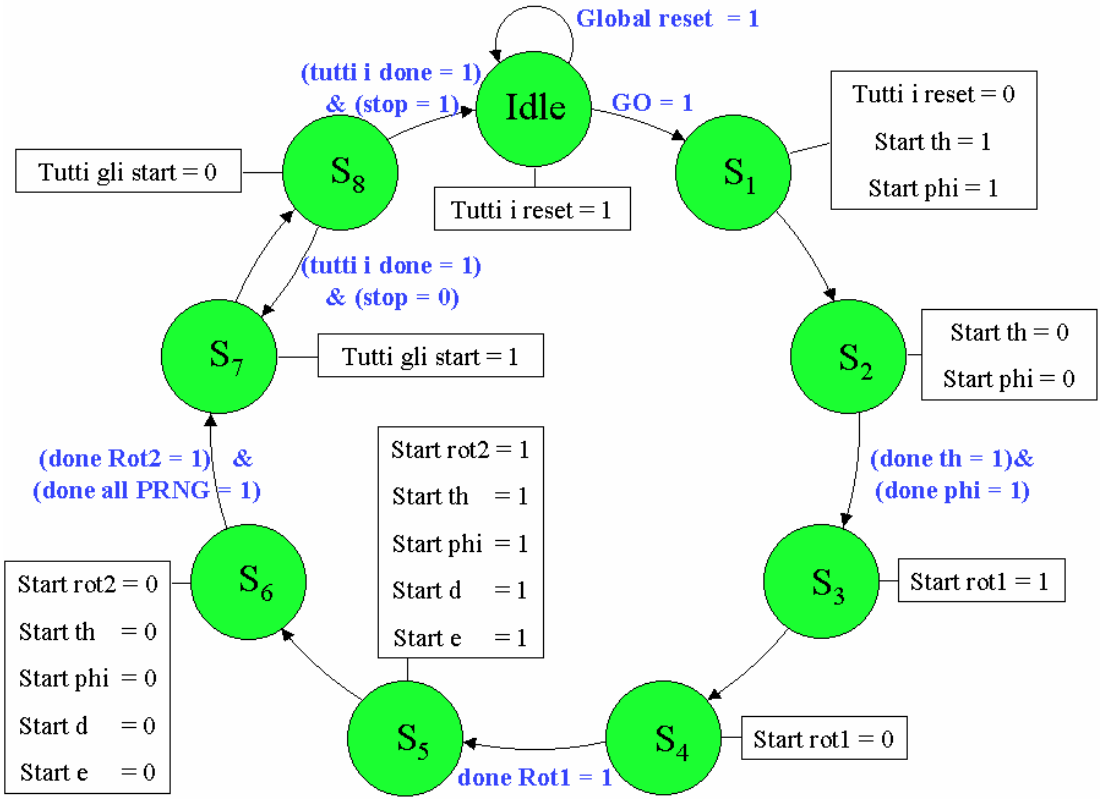

Figura 5.6

Inizialmente la macchina si trova in uno stato di "idle", nel quale sono attivati i reset di tutti i blocchi: in questo stato la macchina non processa alcun fotone ma cancella tutte le informazioni contenute nei registri dei vari blocchi, imposta le condizioni iniziali di alcune variabili e carica i diversi semi nei generatori di numeri pseudo-casuali; lo stato della macchina rimane questo finché il reset globale della macchina rimane attivo e dall'esterno non gli arriva il segnale "go".

Nello stato1 i reset di tutti i blocchi vengono disattivati e si attivano gli start dei generatori di numeri pseudo-casuali Rnd th e Rnd phi i quali, conseguentemente, generano il primo numero.

Al successivo ciclo di clock, lo stato diventa pari a 2: in questo stato di transizione, che incontreremo diverse volte, vengono disattivati gli start di tutti i blocchi (nel caso in questione, ovviamente solo quelli dei generatori di numeri pseudo-casuali); in queste condizioni i blocchi che, nello stato precedente, avevano ricevuto un impulso di start, continuano a elaborare i dati finché non è disponibile il dato processato in

uscita; a questo punto, i blocchi che hanno finito si fermano e inviano alla logica di controllo il proprio segnale di "done".

Quando la logica di controllo riceve un segnale di done dai blocchi Rnd th e Rnd phi passa allo stato 3. In questo stato viene attivato lo start del primo blocco di rotazione, il quale inizia quindi a processare i valori di Θ e Φ ricevuti dai due generatori.

La macchina passa quindi allo stato 4, il quale è un nuovo stato di transizione; lo stato non varia finché la logica di controllo non riceve il segnale di "done" dal primo blocco di rotazione.

Quando lo stato diventa pari a 5, si attiva lo start del secondo blocco di rotazione e contemporaneamente quello di tutti i generatori di numeri casuali: in questo stato, mentre il blocco Rot 2 calcola i nuovi valori di {u, v, w}, gli altri blocchi generano il valore dell'energia persa dal fotone a causa dello scattering (Rnd E), la distanza percorsa prima dell'urto (Rnd d) e i nuovi angoli di scattering per la successiva interazione (Rnd th e Rnd phi).

La macchina passa quindi al nuovo stato di transizione, lo stato 6; da questo passa al successivo solo quando la logica di controllo riceve il segnale di "done" dal blocco Rot2 e del "done" di tutti i generatori di numeri casuali.

Nello stato successivo, lo stato 7, vengono attivati tutti gli start dei vari blocchi: in questo stato, tutte le singole parti della macchina processano i dati che si trovano al loro ingresso.

Da questo stato si passa quindi, al successivo ciclo di clock, allo stato di transizione 8, in cui abbiamo due possibili passaggi di stato: se tutti i blocchi, azionati nello stato precedente, inviano il loro segnale di done e il blocco di stop non verifica la condizione per la quale si deve interrompere il processo del fotone, allora la macchina torna nello stato 7 per simulare un altro passo del fotone, altrimenti termina la sua storia e passa nello stato di idle per iniziare il processo di un nuovo fotone.

5.6 Pipeline.

Il pipeline è una tecnica in cui un calcolo è suddiviso in una sequenza di passaggi o unità funzionali; ad ogni istante di tempo, dopo una fase di caricamento iniziale, ogni unità funzionale produce un nuovo risultato parziale e lo passa allo stadio successivo. In ciascun passaggio della sequenza vengono eseguite più operazioni in parallelo; questa tecnica dunque non velocizza l'esecuzione della singola operazione ma soltanto l'intero processo. I processori basati su questo tipo di architettura sono detti processori vettoriali.

Per come è stata progettata, la nostra macchina rispecchia in parte questa tecnica: infatti, nell'oscillazione tra lo stato 7 e lo stato 8, tutti i blocchi lavorano all'unisono producendo i risultati parziali del calcolo nel medesimo stato. Al contrario delle macchine che realmente sfruttano il pipeline, la nostra macchina non produce un risultato a ogni ciclo di clock ma assume la velocità del blocco più lento; infatti, nell'oscillazione tra lo stato 7 e lo stato 8, quando un blocco ha prodotto il suo risultato parziale non può passarlo al blocco successivo per processare il dato successivo finché anche tutti gli altri blocchi non hanno finito i loro calcoli.

Nel nostro caso reale, ad esempio, i generatori di numeri pseudo-casuali possono produrre un nuovo numero ad ogni ciclo di clock, in quanto il loro processo parziale consiste solo in uno shift dei numeri contenuti nel loro registro, ma sono costretti ad aspettare che il blocco Rot1, il più lento di tutta la macchina, esegua l'algoritmo CORDIC, la radice quadrata di $(1 - w_0^2)$ e le due divisioni per ottenere Sen φ_0 e Cos φ_0; la velocità di esecuzione di Rot1 è di 41 cicli di clock, dunque questo è il numero di cicli necessari a calcolare completamente lo step di un fotone.

Dalle considerazioni fatte, possiamo quindi affermare che la nostra macchina non lavora in pipeline ma, visto il meccanismo di suddivisione del calcolo che è stato realizzato, in una sorta di pseudo-pipeline.

5.7 Schema dettagliato dei singoli blocchi.

Analizziamo ora nel dettaglio come sono realizzati internamente i blocchi che compongono la nostra macchina; non consideriamo i blocchi dei generatori di numeri pseudo-casuali in quanto sono stati descritti approfonditamente nei paragrafi precedenti.

Nella Figura 5.7 è rappresentato lo schema del blocco rot1;

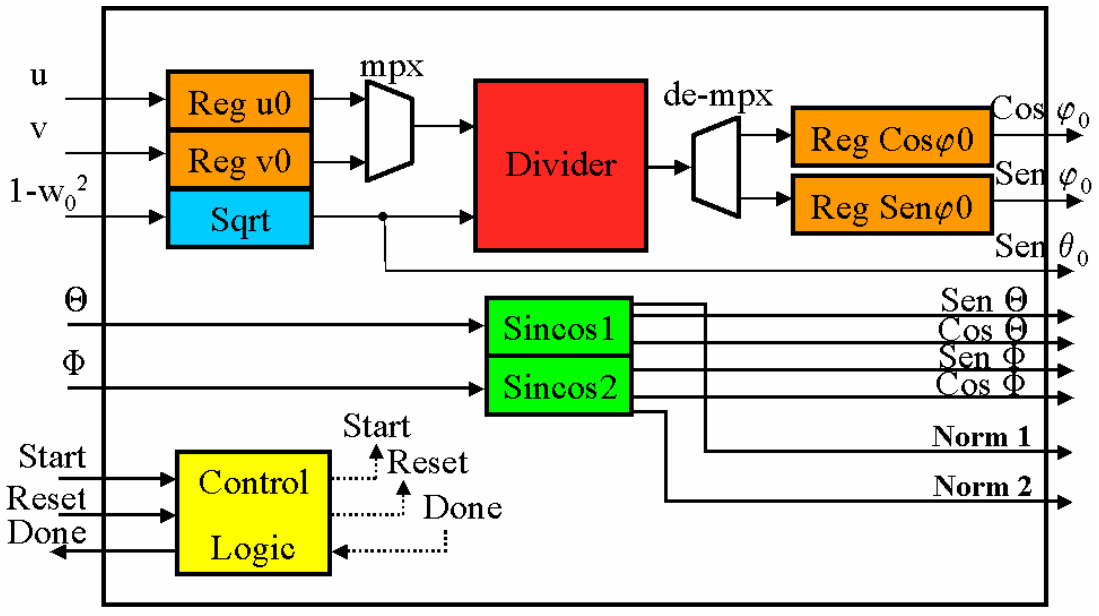

Figura 5.7

Come si può osservare anche all'interno di questo blocco si trova una logica di controllo sequenziale; il motivo della sua presenza è che, mentre il blocco che calcola il Seno e il Coseno di Θ e Φ (Sincos) può lavorare indipendentemente dal resto del blocco, il divisore (Divider) non può eseguire la sua operazione finché il blocco della radice quadrata (Sqrt) non ha esaurito il suo calcolo: è necessario quindi che ci sia un logica che piloti il propagarsi dei segnali all'interno di questo blocco, con una struttura simile a quella della logica che pilota l'intera macchina.

Nella Figura 5.8 possiamo invece vedere lo schema del secondo blocco di rotazione, Rot2:

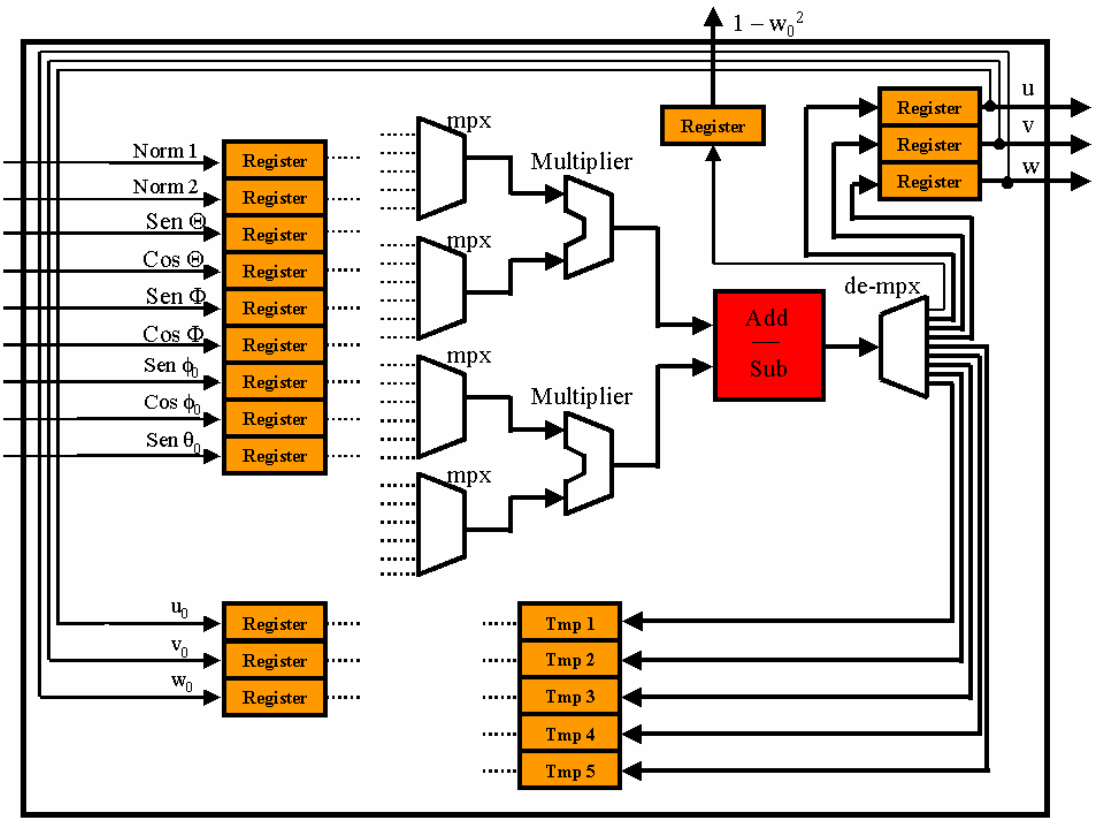

Figura 5.8

Questa parte, come si può osservare, è puramente combinatoria; questo è dovuto al fatto che tutte le operazioni svolte dai singoli blocchi componenti sono eseguite in un solo ciclo di clock per volta, quindi è sufficiente fare gli opportuni collegamenti con dei multiplexer affinché le operazioni siano fatte nella sequenza giusta. Siccome la moltiplicazione tra due numeri a 16 bit dà come risultato un numero a 32 bit, si è scelto, per non aumentare a ogni passaggio la dimensione della parola all'interno della macchina, di considerare come risultato del prodotto solo i 16 bit più significativi; questa approssimazione, che corrisponde alla realizzazione di un "troncamento", è stata preferita all'arrotondamento perché più semplice da implementare.

Il blocco di traslazione è invece rappresentato nella Figura 5.9:

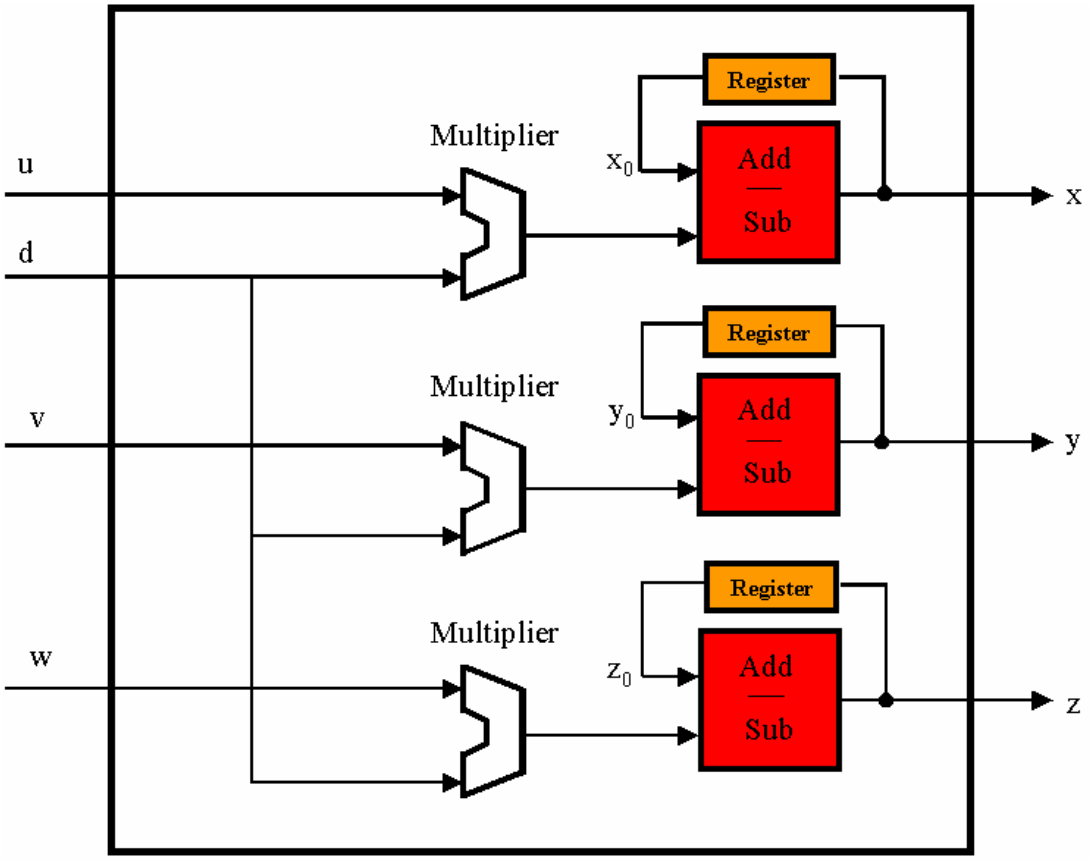

Figura 5.9

Anche questo blocco è puramente combinatorio e contiene tre moltiplicatori a 16 bit molto più lenti dei precedenti: l'algoritmo con cui essi infatti calcolano il prodotto dei due numeri in ingresso richiede un numero di cicli, che dipende dalla dimensione delle parole in ingresso, che può essere anche pari a 16. Questo è stato fatto per non aumentare eccessivamente il numero delle CLB occupate da tutti questi moltiplicatori ed è stato possibile grazie al fatto che la velocità del blocco più lento è di 41 cicli; non ha, infatti, senso avere un blocco velocissimo se poi questo è costretto ad aspettare il blocco più lento.

Il blocco di stop è costituito, infine, da un semplice sottrattore, per calcolare la nuova energia del fotone dopo lo scattering, e di un comparatore a 16 bit, per verificare se l'energia è minore dell'energia di soglia per l'interazione.

Capitolo 6

Confronto tra software e hardware

6.1 Introduzione.

Nei capitoli precedenti abbiamo descritto in maniera particolareggiata l'architettura degli algoritmi sviluppati in software e hardware. Le due realizzazioni sono profondamente diverse tra loro: nel linguaggio C, infatti, le istruzioni vengono eseguite una di seguito all'altra nell'ordine in cui sono scritte mentre in Verilog più istruzioni possono essere eseguite nello stesso ciclo di clock. La frequenza con cui poi vengono eseguite le istruzioni è molto diversa nei due casi: infatti, sebbene il software in C sia stato compilato su un Pentium™ III a 750 MHz, non si può considerare questa come la frequenza di esecuzione vera del programma in quanto essa rappresenta la frequenza del processore, il quale compie anche altre operazioni durante l'esecuzione del programma, come la gestione della scheda video. La frequenza con cui invece ipotizziamo di lavorare con la scheda su cui sarà montata la Spartan™ II è di circa 100 MHz, con possibilità di miglioramenti nelle successive versioni.

Il confronto tra il tempo di esecuzione dei due meccanismi è dunque molto complicato a causa di queste diversità; inoltre, a causa dell'impossibilità di lavorare con l'oggetto hardware reale, abbiamo fatto uso, per la misura del tempo di esecuzione

del programma in Verilog, dell'analizzatore di stati logici incluso nel software Foundation: tramite questa simulazione fisica del progetto, è stato possibile configurare la frequenza del clock al valore desiderato, analizzare la corretta esecuzione dell'algoritmo e verificare a ogni step la presenza di eventuali temporizzazioni errate nella successione dei segnali.

Lo studio dei due differenti tempi di esecuzione, che ora discuteremo, ci ha permesso di verificare le nostre aspettative: la realizzazione hardware del processo è, infatti, risultata più veloce del suo analogo software di circa due ordini di grandezza.

6.2 Misura del tempo di esecuzione del software.

In molti casi è importante sapere qual'é il tempo di esecuzione di un programma e, in particolare, la ripartizione dei tempi di esecuzione delle singole istruzioni che lo compongono: questa operazione è detta "profiling". In un profiling non è solo il tempo di esecuzione che viene calcolato, ma anche lo spazio occupato in memoria dal progetto. Esistono diverse tecniche per eseguire questa operazione.

Nel nostro caso, la misura del tempo di esecuzione del software è stata effettuata nelle migliori condizioni in cui è stato possibile per noi lavorare: a nostra disposizione avevamo infatti un calcolatore con un processore Pentium™ III a 750 Mhz. La frequenza del clock di questo processore è pari alla metà della massima disponibile negli ultimi modelli attualmente in commercio, che è di 1,5 GHz; questo fatto vedremo che sarà di notevole importanza nelle analisi conclusiva dei risultati.

Per effettuare questa misura abbiamo utilizzato una funzione di sistema di Visual C++ nota come clock_t; questa funzione restituisce il numero di cicli di clock impiegati dal processore per eseguire un certo processo e una costante che rappresenta il numero di cicli di clock che avvengono in un secondo. Tramite questa funzione abbiamo misurato l'ammontare del tempo macchina realmente usato per l'esecuzione del processo del ciclo di for nel quale vengono calcolate tutte le coordinate in cui il

fotone subisce uno scattering e l'energia rimasta dopo ogni interazione. Il tempo, in secondi, è calcolato dividendo il numero di cicli di clock per il numero di cicli di clock al secondo.

Per verificare che la misura fornita da questa funzione fosse realmente quella del numero di cicli di clock impiegati per eseguire il processo da noi selezionato abbiamo eseguito diverse prove aumentando il numero dei fotoni da processare, ricavando la seguente tabella di dati:

Numero di fotoni	Tempo di esecuzione (sec)
30000	3.13
60000	6.21
100000	10.32
200000	20.52

da cui si può osservare che l'andamento del tempo di esecuzione misurato ha un andamento lineare rispetto al numero di fotoni processati, come rappresentato nella Figura 6.1.

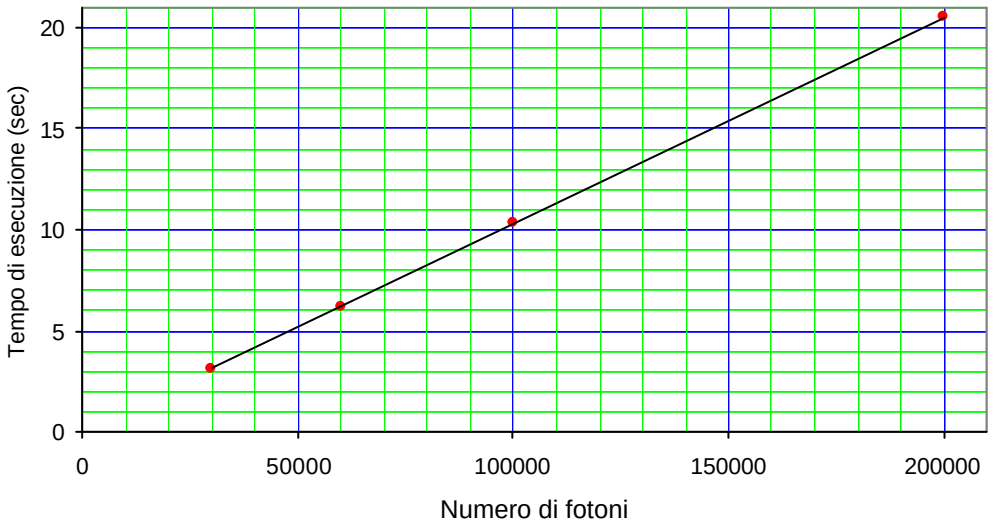

Figura 6.1

Dato che si voleva misurare il tempo necessario al processore per simulare la storia di un solo fotone, visto che la precisione offerta dalla funzione clock_t era del centesimo di secondo mentre il tempo di esecuzione che ci aspettavamo di misurare era dell'ordine del microsecondo, abbiamo ricavato questa misura dividendo il tempo di esecuzione necessario per processare N fotoni per il numero stesso N di fotoni, ottenendo i seguenti valori:

Tempo necessario al processo di un fotone
$1.04 \cdot 10^2$ μsec
$1.03 \cdot 10^2$ μsec
$1.03 \cdot 10^2$ μsec
$1.03 \cdot 10^2$ μsec

In questo modo abbiamo anche reso trascurabile l'errore dovuto al fatto che nella misura fornita da questa funzione è compreso anche il tempo di chiamata della funzione stessa: quando, infatti, il tempo di esecuzione è molto più grande del tempo di chiamata della funzione, possiamo considerare il valore di quest'ultima grandezza trascurabile e quindi assumere come valore vero del tempo di esecuzione la misura così ottenuta, che è stata di 100 μsec con un margine di errore dell'ordine del microsecondo.

6.3 Misura del tempo di esecuzione dell'hardware.

Per eseguire la misura del tempo di esecuzione dell'algoritmo sviluppato in Verilog abbiamo utilizzato il simulatore fisico fornito dalla Xilinx. Questo simulatore permette di settare come si desidera il clock con cui si vuole far andare la FPGA e analizza l'andamento di tutti i segnali logici tenendo conto dei ritardi reali a cui vanno incontro i segnali nel propagarsi attraverso le varie porte logiche configurate

all'interno della FPGA, tenendo in considerazione anche le caratteristiche specifiche del tipo di FPGA su cui si intende caricare il bitstream, nel nostro caso la Spartan II.

Con questo strumento è stato possibile verificare la correttezza dei numeri prodotti dalla macchina, per confronto con quelli prodotti dal programma in Visual C++, e correggere diversi problemi di temporizzazione dei segnali: infatti, a causa del gran numero di porte logiche configurate all'interno della FPGA, sono stati riscontrati diversi ritardi nella propagazione dei segnali a causa dei quali non si verificavano più le condizioni necessarie alle transizioni da uno stato ad un altro.

Nella Figura 6.2 è rappresentata la schermata del simulatore fisico di Foundation:

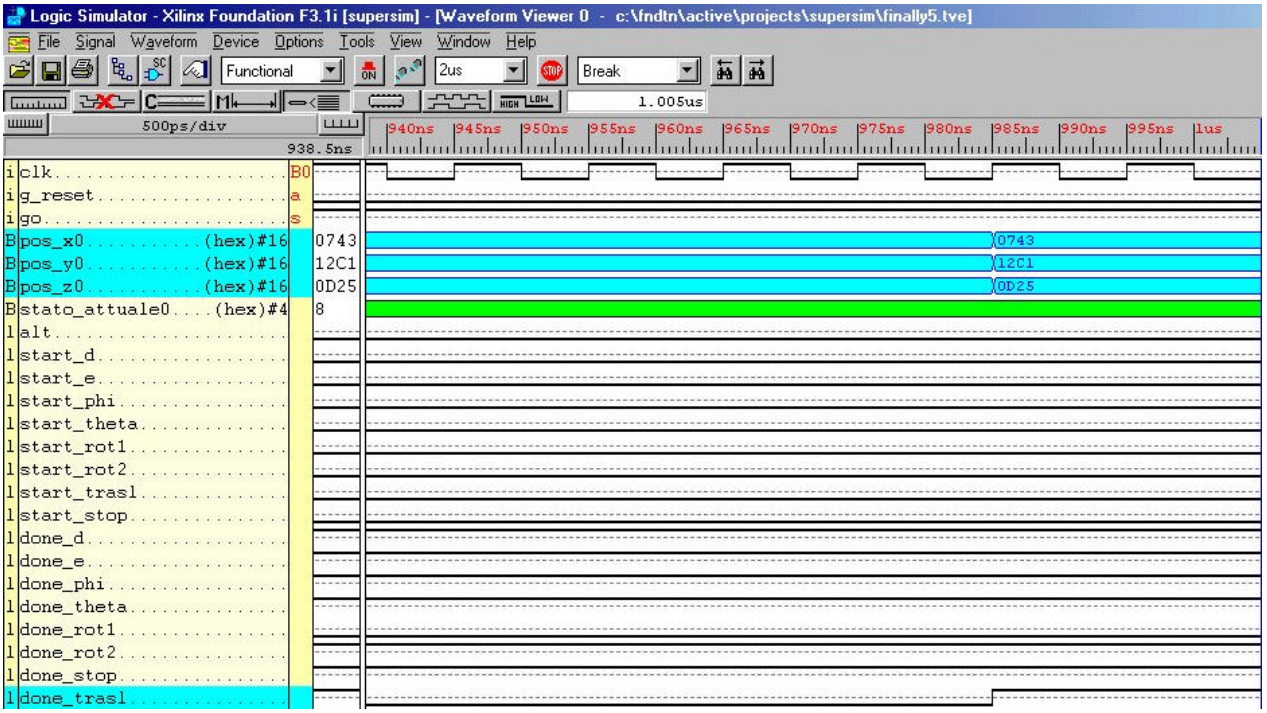

Figura 6.2

Come si può osservare il periodo scelto per il clock è stato di 10 ns, equivalente a una frequenza di 100 MHz, e i dati delle coordinate in cui il fotone subisce la prima interazione vengono prodotti, contemporaneamente al sollevarsi del segnale di "done" del blocco di traslazione (trasl), dopo un tempo di poco inferiore al microsecondo.

Conclusioni

Come mostrato nel corso di questo lavoro, si può affermare che è possibile trarre un guadagno reale, in termini di efficienza, dall'approccio "hardware" nella risoluzione del problema trattato.

Dall'analisi dei dati presentati, infatti, si può vedere che il guadagno nel tempo di esecuzione tra la realizzazione software e quella hardware del metodo Monte Carlo da noi realizzato è stato di un fattore 10^2. È ragionevolmente pronosticabile che questo risultato migliori nelle successive versioni del programma in Verilog; il nostro, infatti, è solamente un prototipo con molti limiti che vuole mostrare con chiarezza il guadagno di efficienza che si ottiene sviluppando algoritmi tipici del linguaggio software con una nuova mentalità, quella del programmatore di hardware.

I limiti stringenti a cui siamo stati legati nella realizzazione del progetto erano per lo più dovuti alla necessità di dover sempre trovare un compromesso tra la velocità con cui si voleva far lavorare i singoli blocchi, lo spazio occupato da questi nelle slices della Spartan II e la dimensione della parola che viene trattata dalla macchina: con la velocità (intesa come numero di cicli di clock necessari ad eseguire un operazione) infatti aumenta anche lo spazio occupato all'interno della FPGA da quel blocco, e così succede pure per l'aumentare della precisione con cui noi abbiamo scelto di rappresentare le quantità da processare.

Si potrebbe pensare che il guadagno da noi ottenuto sarà probabilmente cancellato tra qualche anno: infatti, con il continuo sviluppo della tecnologia dei microprocessori, tra non molto tempo sarà possibile disporre, a prezzi accessibili da un utente comune, di un microprocessore che possa lavorare a frequenze molto elevate. Un andamento analogo a questa crescita della frequenza di lavoro dei microprocessori sarà però ipotizzabile anche per quanto riguarda la velocità di lavoro della FPGA: è ragionevole ritenere quindi che il guadagno, in termini di efficienza, non sarà appiattito in un futuro prossimo semplicemente dalle evoluzioni della tecnologia.

Come prospettive per il futuro si prevede, quindi, di realizzare un algoritmo in hardware su dei modelli fisici più realistici di quello da noi considerato, di ottimizzare l'algoritmo Verilog costruito in questo lavoro, per semplificare il codice, di indirizzare l'impostazione dello schema della macchina verso il pipeline, di ridimensionare l'elevato spazio occupato dal nostro prototipo all'interno della FPGA e di realizzare la scheda di interfaccia per il collegamento di una o più Spartan II con il bus PCI.

Appendice A

Listati del simulatore Monte Carlo in C++

```
//   monte.cpp :     Defines the entry point for the console application.
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "func.h"
#include "const.h"
#include "data.h"

int main (int argc, char* argv[])
{
    /********************* Initialization **********************/
    int i;
    FILE* outFile;
    double distance, theta, phi;
    Position pos[MAX_STEPS];
    Direction direct[MAX_STEPS];
    double energy = MAX_ENERGY;
    srand((unsigned)time( NULL ));                    /* Random seed */
    pos[0].x = 0;
    pos[0].y = 0;
    pos[0].z = 0;
    direct[0].phi = GetRndPhi();
    direct[0].theta = GetRndTheta();
    CosinesFromAngles(&(direct[0]));
    /****************** Open file for output ********************/
    outFile = fopen( "mcOut.txt", "w" );
    fprintf(outFile, "x\t\ty\t\tz\t\tE\t\t\n");
    /******************* Print initial position *****************/
    fprintf(outFile, "%f     %f     %f     %f\n", pos[0].x,  pos[0].y, pos[0].z, energy);
    /*************** Monte Carlo simulation steps  **************/
    for(i = 0; i < MAX_STEPS; i++)
    {
        distance = GetRndDistance();                  /* Travel distance */
        Traslation(&(pos[i]), &(pos[i+1]), &(direct[i]), distance);
        fprintf(outFile, "%f     %f     %f     %f\n", pos[i+1].x,
        pos[i+1].y, pos[i+1].z, energy);
        energy -= GetRndEnergy();                     /* Decrease energy */
        if(energy < ENERGY_CUTOFF) break;  /* Stop if E < cutoff */
        theta = GetRndTheta();            /* Random theta and phi angles */
        phi = GetRndPhi();
        Rotation(&(direct[i]), &(direct[i+1]), theta, phi);     /* Rotation */
    }
```

```
    fprintf(outFile, "\nNumber of steps = %d\n",i);
    fclose(outFile);
    return 0;
}



// Functions definitions file
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "const.h"
#include "func.h"
/******************** Random energy *******************************/
double GetRndEnergy()
{
 return MAX_ENERGY_LOST * MAX_ENERGY * (double)(rand())/RAND_MAX;
}
/******************** Random distance *****************************/
double GetRndDistance()
{
 return MAX_DISTANCE * (double)(rand())/RAND_MAX;
}
/******************** Random theta angle **************************/
double GetRndTheta()
{
 return (double)(PI*rand())/RAND_MAX;
}
/******************** Random phi angle ****************************/
double GetRndPhi()
{
 return (double)(2*PI*rand())/RAND_MAX;
}
/******************** Traslation **********************************/
void Traslation(Position* initPos, Position* finalPos, Direction* direct, double distance)
{
 finalPos->x = initPos->x + distance*direct->u;
 finalPos->y = initPos->y + distance*direct->v;
 finalPos->z = initPos->z + distance*direct->w;
}
/******************** Rotation ************************************/
void Rotation(Direction* initDirect, Direction* finalDirect, double theta, double phi)
{
 finalDirect->u = cos(theta)*initDirect->u + sin(theta) * (cos(phi) * cos(initDirect->phi) *initDirect->w –
                  sin(phi) * sin(initDirect->phi));
 finalDirect->v = cos(theta)*initDirect->v + sin(theta) * (cos(phi) * sin(initDirect->phi) *initDirect->w +
                  sin(phi) * cos(initDirect->phi));
 finalDirect->w = cos(theta)*initDirect->w - sin(theta) * cos(phi) * sin(initDirect->theta);
 AnglesFromCosines(finalDirect);
}
/****** Theta and phi angles from direction cosines values *******/
void AnglesFromCosines(Direction* direct)
{
 double sinTheta, cosPhi, sinPhi;
 direct->theta = acos(direct->w);
 sinTheta = sqrt(1 - direct->w * direct->w);
 cosPhi = direct->u / sinTheta;
 sinPhi = direct->v / sinTheta;
    if(sinPhi >= 0 )
        direct->phi = acos(cosPhi);
    else if (sinPhi < 0 )
        direct->phi = 2*PI - acos(cosPhi);
}
/********** Direction cosines values from theta and phi **********/
void CosinesFromAngles(Direction* direct)
{
 direct->u = sin(direct->theta) * cos(direct->phi);
 direct->v = sin(direct->theta) * sin(direct->phi);
 direct->w = cos(direct->theta);
}
```

```
// Constants definition file header
#ifndef __CONST_H__
#define __CONST_H__
const double PI = 3.1415926535;
const int MAX_STEPS = 1000;                    /* Max number of MC steps*/
const double MAX_ENERGY = 100;                 /* In Kev */
const double MAX_ENERGY_LOST = 0.1;            /* In % */
const double ENERGY_CUTOFF = 1;                /* In Kev */
const double MAX_DISTANCE = 7;                 /* in mm */
#endif /* __CONST_H__ */
```

```
// Data declaration file header
#ifndef __DATA_H__
#define __DATA_H__
struct Position            /* Photon position */
{
  double x;
  double y;
  double z;
};
struct Direction       /* Photon propagation direction */
{
  double theta;
  double phi;
  double u;
  double v;
  double w;
};
#endif /* __DATA_H__ */
```

```
// Functions declaration file header
#ifndef __FUNC_H__
#define __FUNC_H__
#include "data.h"
double GetRndEnergy();          /* Return random number with "energy" distribution*/
double GetRndDistance();        /* Return random number with "distance" distribution */
double GetRndTheta();           /* Return random number with theta" distribution */
double GetRndPhi();             /* Return random number with "phi" distribution */
void Traslation(Position* initPos, Position* finalPos, Direction* direct, double distance); /* Perform traslation from initial
                                                                                              position inPos to final
                                                                                              position outPos, in the
                                                                                              direction indicated by
                                                                                              "direct" direction;
                                                                                              "distance" is the translation
                                                                                              distance */
void Rotation(Direction* initDirect, Direction* finalDirect, double theta, double phi);/* Perform rotation from inDirect
                                                                                         direction to outDirect direction;
                                                                                         theta and phi are rotation angles */
void AnglesFromCosines(Direction* direct);          /* Calculates theta and phi angles from direction cosines values */
void CosinesFromAngles(Direction* direct);          /* Calculates direction cosines values from theta and phi angles */
#endif /* __FUNC_H__ */
```

Appendice B

Listati del simulatore Monte Carlo in Verilog

```
                                //MAIN DELL'INTERA MACCHINA//

module main (clk,g_reset,seme_th,seme_phi,seme_d,seme_e,go,pos_x,pos_y,pos_z);

`include "const.v"

parameter dim_state = 4;

input clk,g_reset,go;
input [15:0] seme_th,seme_phi,seme_d,seme_e;
output[dim_pos - 1:0] pos_x,pos_y,pos_z;

reg    [dim_pos - 1:0] pos_x,pos_y,pos_z;
reg    [dim_state - 1:0] stato_attuale, stato_successivo;
reg    reset_trasl,reset_rot1,reset_rot2,reset_d,reset_theta,reset_phi,reset_stop,
       reset_e;
reg    start_trasl,start_rot1,start_rot2,start_d,start_theta,start_phi,start_stop,
       start_e;

wire   15:0] seme_th,seme_phi,seme_d,seme_e;
wire   reset_trasl_or,reset_rot1_or,reset_rot2_or,reset_d_or,reset_theta_or,
       reset_phi_or,reset_stop_or,reset_e_or;
wire   [dim_d - 1:0] d;
wire   [dim_theta - 1:0] THETA;
wire   [dim_phi - 1:0] PHI;
wire   [dim_cosdir - 1:0]cosdir_u,cosdir_v,cosdir_w;
wire   [dim_cosdir - 1:0]u, v, w, uno_meno_w0quadro;
wire   [dim_sincos - 1:0]Sen_THETA, Cos_THETA, Sen_PHI, Cos_PHI;
wire   [dim_sincos - 2:0]Sen_phi0, Cos_phi0, Sen_theta0;
wire   [dim_sincos - 2:0]norm1, norm2;
wire   done_trasl,done_rot1,done_rot2,done_d,done_theta,done_phi,done_stop,done_e;
wire   alt;
wire   done_tp,done_tprde,done_all,done_allPRNG;
wire   dim_e - 1:0] e;

assign done_tp      = done_theta & done_phi;
assign done_allPRNG = done_theta & done_phi & done_d & done_e;
assign done_tprde   = done_theta & done_phi & done_rot1 & done_rot2 & done_d & done_e;
assign done_all     = done_theta & done_phi & done_rot1 & done_rot2 & done_d & done_e &
                      done_trasl & done_stop;

assign reset_trasl_or = reset_trasl | g_reset;
assign reset_rot1_or  = reset_rot1  | g_reset;
assign reset_rot2_or  = reset_rot2  | g_reset;
```

```
assign reset_d_or     = reset_d     | g_reset;
assign reset_theta_or = reset_theta | g_reset;
assign reset_phi_or   = reset_phi   | g_reset;
assign reset_stop_or  = reset_stop  | g_reset;
assign reset_e_or     = reset_e     | g_reset;

rnd_d       g1 (seme_d,clk,reset_d_or,start_d,d,done_d);
rnd_th      g2 (seme_th,clk,reset_theta_or,start_theta,THETA,done_theta);
rnd_phi     g3 (seme_phi,clk,reset_phi_or,start_phi,PHI,done_phi);
main_Rot1   g4 (clk, start_rot1, reset_rot1_or, u, v, uno_meno_w0quadro, THETA, PHI,
                Sen_THETA, Cos_THETA, Sen_PHI, Cos_PHI, norm1, norm2,Sen_phi0,Cos_phi0,
                Sen_theta0, done_rot1);
Rot2main    g5 (clk, start_rot2, reset_rot2_or,Sen_THETA, Cos_THETA, Sen_PHI, Cos_PHI,
                Sen_theta0, Sen_phi0, Cos_phi0, norm1, norm2,u, v, w,uno_meno_w0quadro,
                done_rot2);
trasl       g6 (clk, reset_trasl_or, start_trasl, d, u, v, w,pos_x, pos_y, pos_z,
                done_trasl);
rnd_e       g7 (seme_e,clk,reset_e_or,start_e,e,done_e);
stop        g8 (clk,reset_stop_or,start_stop,e,alt,done_stop);

always @ (go or stato_attuale or done_tp)
 begin
   stato_successivo = 0;

    case (stato_attuale)

      0 :
      begin
        reset_trasl = 1;
        reset_rot1  = 1;
        reset_rot2  = 1;
        reset_stop  = 1;
        reset_theta = 1;
        reset_phi   = 1;
        reset_d     = 1;
        reset_e     = 1;

        start_trasl = 0;
        start_rot1  = 0;
        start_rot2  = 0;
        start_stop  = 0;
        start_theta = 1;//  I GENERATORI DI NUMERI CASUALI CARICANO IL SEME
        start_phi   = 1;//    FINCHE' LA MACCHINA RIMANE IN QUESTO STATO
        start_d     = 0;
        start_e     = 0;

        if (go) stato_successivo = 1;
        else    stato_successivo = 0;
      end

      1 :
      begin
        reset_trasl = 0;
        reset_rot1  = 0;
        reset_rot2  = 0;
        reset_stop  = 0;
        reset_theta = 0;
        reset_phi   = 0;
        reset_d     = 0;
        reset_e     = 0;

        start_trasl = 0;
        start_rot1  = 0;
        start_rot2  = 0;
        start_stop  = 0;
        start_theta = 1; //RND_THETA INIZIA LA GENERAZIONE DEI NUMERI CASUALI
        start_phi   = 1; // RND_PHI  INIZIA LA GENERAZIONE DEI NUMERI CASUALI
        start_d     = 0;
        start_e     = 0;


        stato_successivo  = 2;
      end
```

```
2 :
begin
 reset_trasl = 0;
 reset_rot1  = 0;
 reset_rot2  = 0;
 reset_stop  = 0;
 reset_theta = 0;
 reset_phi   = 0;
 reset_d     = 0;
 reset_e     = 0;

 start_trasl = 0;
 start_rot1  = 0;
 start_rot2  = 0;
 start_stop  = 0;
 start_theta = 0;
 start_phi   = 0;
 start_d     = 0;
 start_e     = 0;

 if (done_tp) stato_successivo = 3;
 else         stato_successivo = 2;
end

3 :
begin
 reset_trasl = 0;
 reset_rot1  = 0;
 reset_rot2  = 0;
 reset_stop  = 0;
 reset_theta = 0;
 reset_phi   = 0;
 reset_d     = 0;
 reset_e     = 0;

 start_trasl = 0;
 start_rot1  = 1;//IL BLOCCO ROT1 INIZIA LA PRIMA PARTE DELLA ROTAZIONE
 start_rot2  = 0;
 start_stop  = 0;
 start_theta = 0;
 start_phi   = 0;
 start_d     = 0;
 start_e     = 0;

 stato_successivo  = 4;
end

4 :
begin
 reset_trasl = 0;
 reset_rot1  = 0;
 reset_rot2  = 0;
 reset_stop  = 0;
 reset_theta = 0;
 reset_phi   = 0;
 reset_d     = 0;
 reset_e     = 0;

 start_trasl = 0;
 start_rot1  = 0;
 start_rot2  = 0;
 start_stop  = 0;
 start_theta = 0;
 start_phi   = 0;
 start_d     = 0;
 start_e     = 0;

 if (done_rot1) stato_successivo = 5;
 else           stato_successivo = 4;
end
```

```
5 :
begin
  reset_trasl = 0;
  reset_rot1  = 0;
  reset_rot2  = 0;
  reset_stop  = 0;
  reset_theta = 0;
  reset_phi   = 0;
  reset_d     = 0;
  reset_e     = 0;

  start_trasl = 0;
  start_rot1  = 0;
  start_rot2  = 1; // IL BLOCCO ROT2 INIZIA LA SECONDA PARTE DELLA ROTAZIONE
  start_stop  = 0;
  start_theta = 1; // RND_theta GENERA UN NUOVO NUMERO CASUALE
  start_phi   = 1; // RND_phi   GENERA UN NUOVO NUMERO CASUALE
  start_d     = 1; // RND_D INIZIA LA GENERAZIONE DEI NUMERI CASUALI
  start_e     = 1; // RND_E INIZIA LA GENERAZIONE DEI NUMERI CASUALI

  stato_successivo  = 6;
end

6 :
begin
  reset_trasl = 0;
  reset_rot1  = 0;
  reset_rot2  = 0;
  reset_stop  = 0;
  reset_theta = 0;
  reset_phi   = 0;
  reset_d     = 0;
  reset_e     = 0;

  start_trasl = 0;
  start_rot1  = 0;
  start_rot2  = 0;
  start_stop  = 0;
  start_theta = 0;
  start_phi   = 0;
  start_d     = 0;
  start_e     = 0;

  if (done_rot2 & done_allPRNG) stato_successivo = 7;
  else                          stato_successivo = 6;
end

7 :
begin
  reset_trasl = 0;
  reset_rot1  = 0;
  reset_rot2  = 0;

  reset_stop  = 0;
  reset_theta = 0;
  reset_phi   = 0;
  reset_d     = 0;
  reset_e     = 0;
                              // ADESSO TUTTE LE MACCHINE LAVORANO IN PSEUDO-PIPELINE
  start_theta = 1;
  start_phi   = 1;
  start_rot1  = 1;
  start_rot2  = 1;
  start_d     = 1;
  start_e     = 1;
  start_trasl = 1;
  start_stop  = 1;

  stato_successivo  = 8;
end
```

```
      8 :
      begin
        reset_trasl = 0;
        reset_rot1  = 0;
        reset_rot2  = 0;
        reset_stop  = 0;
        reset_theta = 0;
        reset_phi   = 0;
        reset_d     = 0;
        reset_e     = 0;

        start_theta = 0;
        start_phi   = 0;
        start_rot1  = 0;
        start_rot2  = 0;
        start_d     = 0;
        start_e     = 0;
        start_trasl = 0;
        start_stop  = 0;

        if (done_all)
          begin
            if (~alt) stato_successivo  = 7;
            else      stato_successivo  = 0;
          end
        else          stato_successivo  = 8;
      end

    endcase
  end

always @ (posedge clk or posedge g_reset)
  if (g_reset)
    stato_attuale <= 0;
  else
    stato_attuale <= stato_successivo;

endmodule


                    // MAIN DEL GENERATORE DI NUMERI CASUALI //

module mainPRNG (reset, seme, clk, loadseed, sig, out);

input reset, clk, loadseed, sig;
input [15:0] seme;
output[15:0] out;

wire [15:0] seme;
wire [15:0] out;
wire [3:0] cs;
wire muxtoshift, in0, reset;

mux           mux0        (in0, loadseed, sig, muxtoshift);
shiftreg_16   shiftreg0   (reset, seme, muxtoshift, out, clk, cs);
csor_4in      xor0        (cs, in0);

endmodule

                                  // MULTIPLEXER //

module mux (in0, in1, sig, out);

input  in0, in1, sig;
output out;

reg  out;

always @ (sig)
  begin
    case (sig)
```

```
    0 : out = in0;
    1 : out = in1;
    default : out = 'bx;
   endcase
  end
endmodule

                              // SHIFT REGISTER A 16 BIT //

module shiftreg_16 (reset, seme, in, out, clk, cs);

input reset, in, clk;
input [15:0] seme;
output [15:0] out;
output [3:0] cs;

reg [15:0] out;
wire[15:0] seme;

assign cs[0] = out[3];
assign cs[1] = out[12];
assign cs[2] = out[14];
assign cs[3] = out[15];

always @ (negedge clk)
  begin
    if (reset)
      out = seme;
    else
      begin
        out = out << 1;
        out[0] = in;
      end
  end
endmodule

                                  // XOR A 4 INGRESSI //

module csor_4in(cs, in0);

input [3:0] cs;
output in0;

wire in0;

assign in0 = (cs[0] | cs[1] | cs[2] | cs[3]) & (!cs[0] | !cs[1] | !cs[2] | !cs[3]);

endmodule

                  // GENERATORE PSEUDO – CASUALE DELLA VARIABILE d //

module rnd_d (seme,clk,reset,start,d,done);

`include "const.v"

input  clk, reset, start;
input  [15:0] seme;
output [dim_d - 1:0] d;
output done;

reg    [dim_d - 1:0] d;
reg    done;
reg    loaded;

reg    loadseed, sig;
reg    [15:0] out;
reg    RNDgo;

wire   clock, reset;
wire   [15:0] seme;

assign clock = clk;

mainPRNG rnd1 (reset, seme, RNDgo, loadseed, sig, out);
```

```
always @ (negedge clk or posedge reset)
  begin
    if (reset)
      begin
        sig      <= 0;       // il mpx carica il seme
        d        <= 0;
        done     <= 0;
        loaded   <= 0;
        loadseed <= 1;
        RNDgo    <= clock;
      end
    else
      begin
        sig <= 0;       // il mpx riceve in ingresso l'uscita di xor4
        if (~loaded)
          begin
          if (start)
            begin
              d       <= out;
              done    <= 0;
              loaded  <= 1;
              RNDgo   <= 0;
            end
          end
        else if (loaded)
          begin
            done    <= 1;
            loaded  <= 0;
            RNDgo   <= 1;
          end
      end
  end
endmodule

                // GENERATORE PSEUDO – CASUALE DELLA VARIABILE theta //

module rnd_th (seme, clk,reset,start,theta,done);

`include "const.v"

input  clk, reset, start;
input  [15:0] seme;
output [dim_theta - 1:0] theta;
output done;

reg    [dim_theta - 1:0] theta;
reg    done;
reg    loaded;

reg loadseed;
reg sig;
reg [15:0] out;
reg RNDgo;

wire clock, reset;
wire [15:0] seme;

assign clock = clk;

mainPRNG rnd4 (reset, seme, RNDgo, loadseed, sig, out);

always @ (negedge clk or posedge reset)
  begin
    if (reset)
      begin
        sig       <= 0;       // il mpx carica il seme se sig = 1!
        theta     <= 0;
        done      <= 0;
        loaded    <= 0;
        loadseed  <= 1;
        RNDgo     <= clock;
      end
    else
      begin
```

```
    sig <= 0;       // il mpx riceve in ingresso l'uscita di xor4
      if (~loaded)
        begin
          if (start)
            begin
              theta       <= out;
              done        <= 0;
              loaded      <= 1;
              RNDgo       <= 0;
            end
        end
      else if (loaded)
        begin
          done    <= 1;
          loaded  <= 0;
          RNDgo   <= 1;
        end
    end
 end
endmodule

                // GENERATORE PSEUDO – CASUALE DELLA VARIABILE phi //

module rnd_phi (seme, clk,reset,start,phi,done);

`include "const.v"

input  clk, reset, start;
input  [15:0] seme;
output [dim_phi - 1:0] phi;
output done;

reg    [dim_theta - 1:0] phi;
reg    done;
reg    loaded;

reg loadseed;
reg sig;
reg [15:0] out;
reg RNDgo;

wire clock, reset;
wire [15:0] seme;

assign clock = clk;

mainPRNG rnd3 (reset, seme, RNDgo, loadseed, sig, out);

always @ (negedge clk or posedge reset)
 begin
   if (reset)
     begin
       sig       <= 0;       // il mpx carica il seme se sig = 1!
       phi       <= 0;
       done      <= 0;
       loaded    <= 0;
       loadseed  <= 1;
       RNDgo     <= clock;
     end
   else
     begin
     sig <= 0;       // il mpx riceve in ingresso l'uscita di xor4
       if (~loaded)
         begin
           if (start)
             begin
               phi         <= out;
               done        <= 0;
               loaded      <= 1;
               RNDgo       <= 0;
             end
         end
       else if (loaded)
         begin
```

```
         done    <= 1;
         loaded  <= 0;
         RNDgo   <= 1;
         end
       end
  end
endmodule

                        // MAIN DEL PRIMO BLOCCO DI ROTAZIONE //

module main_Rot1(clk, start, reset, u, v, uno_meno_w0quadro, THETA, PHI, Sen_THETA,
                 Cos_THETA, Sen_PHI, Cos_PHI, norm1, norm2, Sen_phi0, Cos_phi0,
                 Sen_theta0,done);

`include "const.v"

input clk, start, reset;
input [dim_cosdir - 1: 0] u, v, uno_meno_w0quadro;
input [dim_theta - 1: 0] THETA;
input [dim_phi - 1: 0] PHI;
output done;
output[dim_sincos - 1: 0] Sen_THETA, Cos_THETA, Sen_PHI, Cos_PHI;
output[dim_sincos - 2: 0] Sen_phi0, Cos_phi0, Sen_theta0;
output[dim_sincos - 2: 0] norm1, norm2;

reg done, eps1, eps2;
reg dim_sincos - 1: 0] Sen_THETA, Cos_THETA, Sen_PHI, Cos_PHI;
reg [dim_sincos - 2: 0]Sen_phi0, Cos_phi0, Sen_theta0;
reg start_SC1, reset_SC1, start_SC2, reset_SC2, start_D, reset_D, reset_R;
reg loaded;
reg [dim_cosdir - 1: 0] u0, v0;
reg [dim_sincos - 2: 0] dividendo, quoziente, resto;
reg [1:0] cntr;
reg [dim_sincos - 2: 0] norm1, norm2;

wire Sen_theta0;
wire done_all;
wire done_SC1, done_SC2, done_D1, done_D2, done_R;

assign done_all = done_SC1 & done_SC2 & done_D2;

sincos     SC1   (clk, reset_SC1, start_SC1, THETA , Sen_THETA, Cos_THETA, norm1,
                  done_SC1);
sincos     SC2   (clk, reset_SC2, start_SC2, PHI, Sen_PHI, Cos_PHI, norm2, done_SC2);
sqrtmain    R    (clk, reset_R, uno_meno_w0quadro, Sen_theta0, done_R);
maindiv     D    (clk, start_D, reset_D, dividendo, Sen_theta0, quoziente, resto,
                  done_D1, done_D2);


always @ (negedge clk or posedge reset)
  begin
    if (reset)
      begin
        start_SC1 <= 0;
        start_SC2 <= 0;
        start_D   <= 0;

        reset_SC1 <= 1;
        reset_SC2 <= 1;
        reset_R   <= 1;
        reset_D   <= 1;

        dividendo <=  0;
        Sen_phi0  <=  16'hE4F8;
        Cos_phi0  <=  16'h727C;
        Done      <=  0;
        cntr       =  0;
        loaded    <=  0;
      end
    else begin
      if (~loaded)
        begin
          if (start)
            loaded <= 1;
```

```
  end
else
  begin
    case (cntr)

      0: begin
        u0         <= u;
        v0         <= v;

        start_SC1  <= 1; // I SINCOS CARICANO I VALORI DI THETA E PHI
        start_SC2  <= 1;
        start_D    <= 0;

        reset_SC1  <= 0;
        reset_SC2  <= 0;
        reset_R    <= 0; //QUANDO SI E' ABBASSATO INIZIA A CALCOLARE LA RADICE DI [1-(W0^2)]
        reset_D    <= 0;

        dividendo   <=  0;
        Sen_phi0    <=  Sen_phi0;
        Cos_phi0    <=  Cos_phi0;
        done        <=  0;
        cntr         =  1;
      end

      1: begin
        start_SC1 <= 0; // I SINCOS INIZIANO A ELABORARE THETA E PHI
        start_SC2 <= 0;
        start_D   <= 0;

        reset_SC1 <= 0;
        reset_SC2 <= 0;
        reset_R   <= 0;
        reset_D   <= 0;

        done      <= 0;
          if (done_R)
          begin
            dividendo <= u0;
            eps1      <= u0[dim_cosdir - 1]; // TIENE CONTO DEL SEGNO DI U0

            start_SC1 <= 0;
            start_SC2 <= 0;
            start_D   <= 1; //IL DIVISORE INIZIA A DIVIDERE

            reset_SC1 <= 0;
            reset_SC2 <= 0;
            reset_R   <= 0;
            reset_D   <= 0;

            done      <= 0;
            cntr       = 2;
          end
        else
        cntr          = 1;
      end

      2: begin
        dividendo <= v0;
        eps2      <= v0[dim_cosdir - 1];                                                    //     TI

        done      <= 0;

        start_SC1 <= 0;
        start_SC2 <= 0;
        start_D   <= 0;

        reset_SC1 <= 0;
        reset_SC2 <= 0;
        reset_R   <= 0;
        reset_D   <= 0;
          if (done_D1 & ~done_D2)
            begin
              if (~eps1)
```

```
                Cos_phi0 <= resto;
              else
                Cos_phi0 <= -resto; // FA IL COMPLEMENTO A 2 SE U0 ERA NEGATIVO
             end
            if (done_D2)
             begin
             if (~eps2)
              Sen_phi0 <= resto;
             else
              Sen_phi0 <=-resto;  // FA IL COMPLEMENTO A 2 SE V0 ERA NEGATIVO
             end
            if (done_all)
             begin
              done    <= 1;
              cntr     = 0;
              loaded  <= 0;
             end
         end
        endcase
      end
    end
  end
endmodule

                // MODULO CHE CALCOLA I SENI E I COSENI DAGLI ANGOLI //

module sincos(clk,reset,start,alpha_in,sin_alpha,cos_alpha,norm,done);

`include "const.v"
parameter nMax = 13;

input  clk, reset, start;
input  [dim_alpha - 1:0] alpha_in;
output [dim_sincos - 1: 0] sin_alpha, cos_alpha;
output done;
output [dim_sincos - 2: 0] norm;

wire   [dim_sincos - 1: 0] sin_alpha_in, cos_alpha_in;

reg    [dim_sincos - 1: 0] sin_alpha, cos_alpha;
reg    [dim_alpha - 3:0] alpha;
reg    [dim_alpha - 2:0] sum;
reg    [1:0] zone;
reg    [4:0] count;
reg    done,stop;
reg    [dim_sincos - 2: 0] norm;
reg    loaded;
reg    eps;
reg    [dim_alpha - 3:0] angle[0:nMax];
reg    [dim_alpha - 1:0] prod[0:nMax];

`include "func_16.v"

assign cos_alpha_in = cos_alpha;
assign sin_alpha_in = sin_alpha;

always @(negedge clk or posedge reset) begin
 if(reset) begin
   sin_alpha  <= 0;
   cos_alpha  <= max_cos;
   sum         = 0;
   done       <= 0;
   loaded      = 0;
   count       = 0;
   eps         = 0;
   stop       <= 0;
   sin_const;
 end
 else begin
   if(~loaded) begin
    if(start) begin
      alpha      <= alpha_in[dim_alpha - 3 :0];
      zone       <= alpha_in[dim_alpha - 1 : dim_alpha - 2];
      sin_alpha  <= 0;
```

```
      cos_alpha   <= max_cos;
      sum          = 0;
      done        <= 0;
      stop        <= 0;
      loaded       = 1;
      count        = 0;
      eps          = 0;
    end
   end
   else begin
    if(~stop) begin
      if(~eps) begin
        cos_alpha  <= cos_alpha_in - (sin_alpha_in >> (count));
        sin_alpha  <= sin_alpha_in + (cos_alpha_in >> (count));
        sum         = sum + angle[count];
      end
      else begin
        cos_alpha  <= cos_alpha_in + (sin_alpha_in >> (count));
        sin_alpha  <= sin_alpha_in - (cos_alpha_in >> (count));
        sum         = sum - angle[count];
      end
      count = count + 1;
      if((sum == alpha) | (count > nMax))
        stop  <= 1;
      else if (sum < alpha | sum[dim_alpha - 2] == 1) eps = 0;
      else eps = 1;
      end
    else begin
      case (zone)
        0: begin
          cos_alpha <= cos_alpha;
          sin_alpha <= sin_alpha;
        end
        1: begin
          cos_alpha <= -sin_alpha;
          sin_alpha <= cos_alpha;
        end
        2: begin
          cos_alpha <= -cos_alpha;
          sin_alpha <= -sin_alpha;
        end
        3: begin
          cos_alpha <=  sin_alpha;
          sin_alpha <= -cos_alpha;
        end
      endcase
     norm    <= prod[count - 1];
     done    <= 1;
     stop    <= 0;
     count    = 0;
     loaded   = 0;
     end
   end
  end
end
endmodule

                      // MODULO CHE CALCOLA LA RADICE QUADRATA //

module sqrtmain (C, reset, DIN, DOUT, done);

input C, reset;
input [15:0] DIN;
output[15:0] DOUT;
output done;

reg [15:0] DOUT;
reg done;
reg CE; //Clock enable: se vale 0 il blocco non processa piu' il dato al suo interno
reg [4:0] cntr;
```

```
//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG

sqrt Radice (
             .DIN(DIN),
             .C(C),
             .CE(CE),
             .DOUT(DOUT));

// INST_TAG_END ------ End INSTANTIATION Template ---------


always @ (negedge C)
 begin
   if (reset)
    begin
      cntr  <= 0;
      CE    <= 1;  // Il blocco carica il dato da processare
      done  <= 0;
    end
   else
    begin
      cntr  <= cntr + 1;
      CE    <= 1;
      done  <= 0;
       if (cntr == 17)
         begin
         done <= 1;
         CE   <= 0;
        end
    end
 end

endmodule

// MODULO CREATO DAL CORE-GENERATOR PER LA RADICE QUADRATA //

/*****************************************************************
* This file was created by the Xilinx CORE Generator tool, and     *
* is (c) Xilinx, Inc. 1998, 1999. No part of this file may be      *
* transmitted to any third party (other than intended by Xilinx)   *
* or used without a Xilinx programmable or hardwire device without *
* Xilinx's prior written permission.                               *
*****************************************************************/

// The following line must appear at the top of the file in which
// the core instantiation will be made. Ensure that the translate_off/_on
// compiler directives are correct for your synthesis tool(s)

// Your Verilog compiler/interpreter might require the following
// option or it's equivalent to help locate the Xilinx Core Library
// +incdir+${XILINX}/verilog/src
// Here ${XILINX} refers to the XILINX software installation directory.

//----------- Begin Cut here for LIBRARY inclusion --------// LIB_TAG

// synopsys translate_off

`include "XilinxCoreLib/sqrootVHT.v"

// synopsys translate_on

// LIB_TAG_END ------- End LIBRARY inclusion --------------

// The following code must appear after the module in which it
// is to be instantiated. Ensure that the translate_off/_on compiler
// directives are correct for your synthesis tool(s).

//----------- Begin Cut here for MODULE Declaration -------// MOD_TAG
module sqrt (
             DIN,
             C,
             CE,
             DOUT);
```

```
input [15 : 0] DIN;
input C;
input CE;
output [15 : 0] DOUT;

// synopsys translate_off

SQROOTVHT #(
            16,
            16)
            inst (
                  .DIN(DIN),
                  .C(C),
                  .CE(CE),
                  .DOUT(DOUT));

// synopsys translate_on

endmodule

// MOD_TAG_END ------- End MODULE Declaration -------------

            // MODULO CONTENENTE L'ALGORITMO PER IL CALCOLO DELLA RADICE //

// $Header:
/devl/xcs/repo/env/Databases/ip/src/com/xilinx/ip/sqrt/simulation/sqrootVHT.v,v 1.5
1999/11/16 16:36:44 cc Exp $
// ***************************************************************************
//  Copyright 1998 - Xilinx, Inc.
//  All rights reserved.
// ***************************************************************************
//
//  Description:
//  Verilog model for square-root
//

module SQROOTVHT(DIN, C, CE, DOUT);

parameter INPUT_WIDTH = 8;
parameter OUTPUT_WIDTH = 8;

input [INPUT_WIDTH-1:0] DIN;
input C, CE;
output [OUTPUT_WIDTH-1:0] DOUT;

reg [OUTPUT_WIDTH-1:0] DOUT;
reg oldc;
reg [INPUT_WIDTH-1:0] dividend [0:OUTPUT_WIDTH-1];
reg [2*OUTPUT_WIDTH-1:0] rem, temp;
integer i, width;

initial
begin
  for(i=0; i<OUTPUT_WIDTH; i=i+1)
    dividend[i] = 1'b0;
  width = INPUT_WIDTH % 2 ? INPUT_WIDTH+1 : INPUT_WIDTH;
  DOUT = 1'b0;
end

always @C
begin
  if (((C === 1'bx || C === 1'bz) && CE !== 1'b0) ||
       (C === 1'b1 && oldc === 1'b0 && (CE === 1'bx || CE === 1'bz)))
  begin
     for (i=0; i<OUTPUT_WIDTH; i=i+1)
       dividend[i] = {INPUT_WIDTH{1'bx}};
     DOUT = {OUTPUT_WIDTH{1'bx}};
  end
  else if (C === 1'b1 && oldc === 1'b0 && CE === 1'b1)
  begin
    if (^dividend[OUTPUT_WIDTH-1] === 1'bx)
      DOUT = {OUTPUT_WIDTH{1'bx}};
    else
    begin
```

```
      if (width > INPUT_WIDTH)
      begin
        rem[2*OUTPUT_WIDTH-2:0] = dividend[OUTPUT_WIDTH-1] << (2*OUTPUT_WIDTH –
                                                               width);
        rem[2*OUTPUT_WIDTH-1] = 1'b0;
      end
      else
        rem = dividend[OUTPUT_WIDTH-1] << (2*OUTPUT_WIDTH - width);

      DOUT = {OUTPUT_WIDTH{1'b0}};
      for (i=1; i<=OUTPUT_WIDTH; i=i+1)
      begin
        temp = (4*DOUT + 1) << (2*OUTPUT_WIDTH - (i*2));
        if (temp <= rem)
        begin
          DOUT = 2*DOUT + 1;
          rem = rem-temp;
        end
        else
          DOUT = 2*DOUT;
      end
    end

    for (i=OUTPUT_WIDTH-1; i>0; i=i-1)
      dividend[i] = dividend[i-1];

    dividend[0] = DIN;
  end
  oldc = C;
end
endmodule

                    //MAIN DEL BLOCCO CHE EFFETTUA LA DIVISIONE//

module maindiv (clk, start, reset, DIVIDEND, DIVISOR, QUOT, REMD, done_D1, done_D2);

input  clk, start, reset;
input  [15:0] DIVIDEND, DIVISOR;
output [15:0]QUOT, REMD;
output done_D1, done_D2;

reg [15:0] QUOT, REMD;
reg done_D1, done_D2, loaded;
reg [5:0]count;
wire C;

assign C = clk & loaded;

//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG

div_u_f DIV (
             .DIVIDEND(DIVIDEND),
             .DIVISOR(DIVISOR),
             .QUOT(QUOT),
             .REMD(REMD),
             .C(C));

// INST_TAG_END ------ End INSTANTIATION Template ---------

always @ (posedge clk)
  begin
    if (reset)
      begin
        loaded  <= 0;
        done_D1 <= 0;
        done_D2 <= 0;
        count    = 0;
      end
    else
      if (~loaded)
        begin
          if (start)
            begin
              done_D1 <= 0;
```

```
          done_D2 <= 0;
          loaded  <= 1;
          count    = 0;
        end
      end
    else
      begin
       count = count + 1;
       if ( count == 34 )
       done_D1 <= 1;
       if ( count == 35 )
         begin
          done_D2 <= 1;
          loaded  <= 0;
         end
      end
 end
endmodule
```

```
               // MODULO CREATO DAL CORE-GENERATOR PER LA DIVISIONE //

/*****************************************************************
* This file was created by the Xilinx CORE Generator tool, and     *
* is (c) Xilinx, Inc. 1998, 1999. No part of this file may be      *
* transmitted to any third party (other than intended by Xilinx)   *
* or used without a Xilinx programmable or hardwire device without *
* Xilinx's prior written permission.                               *
*****************************************************************/

// The following line must appear at the top of the file in which
// the core instantiation will be made. Ensure that the translate_off/_on
// compiler directives are correct for your synthesis tool(s)

// Your Verilog compiler/interpreter might require the following
// option or it's equivalent to help locate the Xilinx Core Library
// +incdir+${XILINX}/verilog/src
// Here ${XILINX} refers to the XILINX software installation directory.

//----------- Begin Cut here for LIBRARY inclusion --------// LIB_TAG

// synopsys translate_off

`include "XilinxCoreLib/dividervht.v"

// synopsys translate_on

// LIB_TAG_END ------- End LIBRARY inclusion --------------

// The following code must appear after the module in which it
// is to be instantiated. Ensure that the translate_off/_on compiler
// directives are correct for your synthesis tool(s).

//----------- Begin Cut here for MODULE Declaration -------// MOD_TAG
module div_u_f (
                DIVIDEND,
                DIVISOR,
                QUOT,
                REMD,
                C);

input [15 : 0] DIVIDEND;
input [15 : 0] DIVISOR;
output [15 : 0] QUOT;
output [15 : 0] REMD;
input C;

// synopsys translate_off

DIVIDERVHT #(
             1,
             16,
             16,
             1,
             16,
```

```
            0)
            inst (
                  .DIVIDEND(DIVIDEND),
                  .DIVISOR(DIVISOR),
                  .QUOT(QUOT),
                  .REMD(REMD),
                  .C(C));

// synopsys translate_on

endmodule

// MOD_TAG_END ------- End MODULE Declaration ------------

          // MODULO CONTENENTE L'ALGORITMO PER IL CALCOLO DELLA DIVISIONE //

//  All rights reserved.
// ***************************************************************************
// Checked out 19th Oct 1998
//
//  Description:
//  pipelined Divider
//
//  $Header:
/devl/xcs/repo/env/Databases/ip/src/com/xilinx/ip/sdivider_v1_0/simulation/DIVIDERVHT
.v,v 1.6 2000/01/12 21:52:26 cc Exp $

module DIVIDERVHT (
      DIVIDEND,
      DIVISOR,
      QUOT,
      REMD,
      C
      );

`define true  1'b1
`define false 1'b0
`define TRUE  1'b1
`define FALSE 1'b0


// Parameter declarations

parameter DIVCLK_SEL       = 1;
parameter DIVIDEND_WIDTH   = 8;
parameter DIVISOR_WIDTH    = 8;
parameter FRACTIONAL_B     = 0;
parameter FRACTIONAL_WIDTH = 8;
parameter SIGNED_B         = 0;


// required constants
parameter latency_max =  DIVIDEND_WIDTH + FRACTIONAL_WIDTH + DIVCLK_SEL + 4;
parameter bus_latency =  (FRACTIONAL_B=== 1) ? DIVIDEND_WIDTH + FRACTIONAL_WIDTH :
DIVIDEND_WIDTH;

//Port declarations
input  [(DIVIDEND_WIDTH-1):0]  DIVIDEND;
input  [(DIVISOR_WIDTH-1):0]    DIVISOR;
output [(DIVIDEND_WIDTH-1):0]  QUOT;
output [(FRACTIONAL_WIDTH-1):0] REMD;
input                      C;


// unsigned ranges
integer U_DIVIDEND_WIDTH;
integer U_DIVISOR_WIDTH;
integer u_bus_latency;


reg [(DIVIDEND_WIDTH-1):0]     QUOT;
reg [(FRACTIONAL_WIDTH-1):0]    REMD;
```

```
integer roc_clock_count;
integer dividend_neg,divisor_neg;
integer quotient_neg,remainder_neg;
integer clock_counter;
integer latency;

// arrays to account for latency

reg [DIVIDEND_WIDTH-1:0] quotient_d[latency_max:0];
reg [FRACTIONAL_WIDTH-1:0] remainder_d[latency_max:0];

reg [DIVIDEND_WIDTH-1:0] vdividend;
reg [DIVISOR_WIDTH-1:0] vdivisor;
reg [bus_latency-1:0] vquotient;
reg [FRACTIONAL_WIDTH-1:0] vremainder,vremainder_temp;
reg [DIVIDEND_WIDTH-1:0] vdividend_temp;
reg [DIVISOR_WIDTH:0] vdivisor_temp,vtemp,vtemp_new;
reg restore;

// for loop control

integer i,j;


initial
begin
   // initialise constants
   clock_counter = 0;
   roc_clock_count = 1;
   latency = pi-
pe_depth(DIVIDEND_WIDTH,FRACTIONAL_WIDTH,DIVCLK_SEL,SIGNED_B,FRACTIONAL_B);
   // unsigned values
   if(SIGNED_B === 1) begin
      U_DIVIDEND_WIDTH = DIVIDEND_WIDTH-1;
      U_DIVISOR_WIDTH    = DIVISOR_WIDTH-1;
      if(FRACTIONAL_B=== 1) begin
         u_bus_latency   = bus_latency-2;
      end
      else begin
         u_bus_latency   = bus_latency-1;
      end
   end
   else begin
      U_DIVIDEND_WIDTH = DIVIDEND_WIDTH;
      U_DIVISOR_WIDTH    = DIVISOR_WIDTH;
      u_bus_latency   = bus_latency;

   end

   // zero out at start

   for (i = 0; i < latency; i = i + 1) begin
      quotient_d[i]  = {(DIVIDEND_WIDTH){1'b0}};
      remainder_d[i] = {(FRACTIONAL_WIDTH){1'b0}};
   end

end // initial

always @(posedge C)
begin
   if (C === 1'bx) begin
      // everything set to X
      for (i = 0; i < latency; i = i + 1) begin
         quotient_d[i]   = {(DIVIDEND_WIDTH){1'bx}};
         remainder_d[i]  = {(FRACTIONAL_WIDTH){1'bx}};
      end
   end
   else begin
      if(roc_clock_count != 0) begin
         //take account of reset time
         roc_clock_count = roc_clock_count -1;
      end
      else
```

```
begin
   // increment the clock counter
   clock_counter = clock_counter +1;
end

if ((clock_counter % DIVCLK_SEL) == 0)
   clock_counter = 0;

// main part of the divisor process
// check first that there is no errors on the input
if( ^DIVIDEND === {1'bx} ||
    ^DIVISOR === {1'bx} ) begin
   vquotient = {(bus_latency){1'bx}};
   vremainder = {(FRACTIONAL_WIDTH){1'bx}};
end
else begin
   // initialise variables and signed booleans
   dividend_neg = `FALSE;
   divisor_neg  = `FALSE;
   vdividend     = DIVIDEND;
   vdivisor      = DIVISOR;

   // check whether the dividend is negative and convert to positive
   if(   SIGNED_B === 1 &&
         (vdividend[DIVIDEND_WIDTH -1] === 1'b1)) begin
      dividend_neg = `TRUE;
      vdividend = ~vdividend+1;
   end

   // check whether the divisor is negative and convert to positive
   if(   SIGNED_B === 1 &&
         (vdivisor[DIVISOR_WIDTH -1] === 1'b1)) begin
      divisor_neg = `TRUE;
      vdivisor = ~vdivisor+1;
   end

   // quotient is negative if XOR
   if ((dividend_neg == `TRUE && divisor_neg == `FALSE)
      || (dividend_neg == `FALSE && divisor_neg == `TRUE))
      quotient_neg = `TRUE;
   else
      quotient_neg = `FALSE;

   // remainder is negative if
   if (dividend_neg == `TRUE)
      remainder_neg = `TRUE;
   else
      remainder_neg = `FALSE;


   vdividend_temp[DIVIDEND_WIDTH -1:0]
      = vdividend[DIVIDEND_WIDTH - 1:0];
   vdivisor_temp[DIVISOR_WIDTH -1:0]
      = vdivisor[DIVISOR_WIDTH - 1:0];

   // set extra top bit of the divisor to zero
   vdivisor_temp[DIVISOR_WIDTH] = 1'b0;
   vdivisor_temp[U_DIVISOR_WIDTH] = 1'b0;
   // initialise vtemp to 0
   vtemp = {(DIVISOR_WIDTH){1'b0}};

   // main loop for the quotient remainder calculation
   for (i = 0; i< u_bus_latency; i = i+1) begin
      restore = vtemp[U_DIVISOR_WIDTH];
      // shift the result
      vtemp[DIVISOR_WIDTH : 1] = vtemp[DIVISOR_WIDTH -1 : 0];
      // add in the lowest bit of the dividend
      if ((U_DIVIDEND_WIDTH - 1 -i) >= 0)
         vtemp[0] = vdividend_temp[U_DIVIDEND_WIDTH-1-i];
      else
         vtemp[0] = 1'b0;

      if (restore === 1'b1) begin
         // add vtemp to vdivsor_temp
```

```
            vtemp_new = vtemp + vdivisor_temp;
         end
         else begin
            // subtract the divisor from vtemp
            vtemp_new = vtemp - vdivisor_temp;
         end

         vtemp = vtemp_new;
         vquotient[u_bus_latency -1 -i] = ~(vtemp[U_DIVISOR_WIDTH]);
      end // quotient remainder calculation loop


      // calculate the remainder
      if(FRACTIONAL_B === 0) begin
         if( vtemp[U_DIVISOR_WIDTH] === 1'b1) begin
            // remainder is result plus divisor
            vremainder_temp = vtemp + vdivisor_temp;
         end
         else
            vremainder_temp[DIVISOR_WIDTH-1:0] = vtemp[DIVISOR_WIDTH -1:0];

         vremainder[DIVISOR_WIDTH -1:0] = vremainder_temp[DIVISOR_WIDTH-1:0];
      end // FRACTIONAL_B = 0

      if(SIGNED_B === 1) begin
         vquotient[bus_latency -1] = 1'b0;
         vremainder[FRACTIONAL_WIDTH - 1] = 1'b0;
         if(FRACTIONAL_B === 1) begin
            vquotient[bus_latency - 2] = 1'b0;
         end
      end

      if(remainder_neg == `TRUE)
         vremainder = ~vremainder+1;

   end

end // if anyX

// move the values through the latency array by one stage
for(i = latency -2; i>=0;i = i-1) begin
   quotient_d[i+1] = quotient_d[i];
   remainder_d[i+1] = remainder_d[i];
end


if(FRACTIONAL_B === 1) begin
   if (SIGNED_B === 0) begin
      remainder_d[0] = vquotient[FRACTIONAL_WIDTH-1 : 0];
   end
   else begin
      vremainder[FRACTIONAL_WIDTH-2: 0] = vquotient[FRACTIONAL_WIDTH-2:0];
         if( ^DIVIDEND === {1'bx} ||
               ^DIVISOR === {1'bx} )
            begin
         vremainder[FRACTIONAL_WIDTH-1] = 1'bx;
      end
      else begin
         vremainder[FRACTIONAL_WIDTH-1] = 1'b0;
      end
      if(quotient_neg == `TRUE)
         vremainder = ~vremainder+1;
      remainder_d[0] = vremainder;
   end
end
else begin
   remainder_d[0] = vremainder;
end

// add in the newest results
if(FRACTIONAL_B === 1 && SIGNED_B === 1) begin
   quotient_d[0] = vquotient[bus_latency -2 : bus_latency-1-DIVIDEND_WIDTH];
   if(quotient_neg == `TRUE)
      quotient_d[0] = ~quotient_d[0]+1;
```

```
   end
   else begin
      quotient_d[0] = vquotient[bus_latency -1 : bus_latency-DIVIDEND_WIDTH];
      if(quotient_neg == `TRUE)
         quotient_d[0] = ~quotient_d[0]+1;
   end

   // put the results out to the ports
   QUOT = quotient_d[latency - 1];
   REMD = remainder_d[latency - 1];

end

// function to calculate latency


function integer pipe_depth;
input [31:0] n,f,div_clk,signed_b,fract;
integer temp;
begin
   if(signed_b === 0) begin
      case (div_clk)
         1: temp = n+3;
         default : temp = n+4;
      endcase
   end
   else begin
      case (div_clk)
         1: temp = n-1 + 3 + 2;
         default : temp = n-1 + 4 + div_clk + 1;
      endcase
   end
   if(fract === 0)
      pipe_depth = temp;
   else begin
      if(signed_b === 0)
         pipe_depth = temp + f;
      else
         pipe_depth = temp + f-1;
   end
end
endfunction


endmodule

////////////////////////////END////////////////////////

                //      MAIN DEL SECONDO BLOCCO DI ROTAZIONE      //

module Rot2main (clock, start, reset, Sen_THETA17, Cos_THETA17, Sen_PHI17,
                 Cos_PHI17, Sen_theta0, Sen_phi0, Cos_phi0, norm1, norm2,
                 u, v, w, uno_meno_w0quadro, done);

`include "const.v"

input  start, clock, reset;
input  [dim_sincos - 1:0]Sen_THETA17,Cos_THETA17,Sen_PHI17,Cos_PHI17;
input  [dim_sincos - 2:0]norm1, norm2;
input  [dim_sincos - 2:0]Sen_phi0, Cos_phi0,Sen_theta0;
output [dim_cosdir - 1:0]u, v, w, uno_meno_w0quadro;
output done;

reg[dim_sincos - 1:0] reg__Sen_THETA17,reg__Cos_THETA17,reg__Sen_PHI17,
                      reg__Cos_PHI17;
reg[2*dim_sincos - 2:0]reg__Sen_THETA32, reg__Cos_THETA32,reg__Sen_PHI32,
                       reg__Cos_PHI32;
reg[dim_sincos - 2:0] reg__Sen_THETA,reg__Cos_THETA,reg__Sen_PHI,reg__Cos_PHI;
reg[dim_sincos - 2:0] reg__norm1, reg__norm2;
reg[dim_sincos - 2:0] reg__Sen_phi0, reg__Cos_phi0,reg__Sen_theta0,uno_meno_w0quadro;
reg[2*dim_sincos - 3:0]reg__w0_per_Cos_phi0,reg__w0_per_Sen_phi0,
                       reg__Sen_THETA_per_Cos_PHI, reg__Sen_THETA_per_Sen_PHI,
                       w0quadro, reg__temp1,reg__temp2,reg__temp3,reg__temp4;
reg[dim_sincos - 2:0] reg__temp5,reg__temp6;
```

```
reg[dim_cosdir - 1:0] u, v, w;
reg[dim_cosdir - 1:0] reg__u0, reg__v0, reg__w0;
reg[3:0] cntr;
reg done, loaded;

always @ (negedge clock or posedge reset)
 begin
   if (reset)
    begin
      reg__Sen_THETA_per_Cos_PHI     <= 0;
      reg__Sen_THETA_per_Sen_PHI     <= 0;
      reg__temp1                     <= 0;
      reg__temp2                     <= 0;
      reg__temp3                     <= 0;
      reg__temp4                     <= 0;
      reg__temp5                     <= 0;
      reg__temp6                     <= 0;
      reg__Sen_THETA17               <= 0;
      reg__Cos_THETA17               <= 0;
      reg__Sen_PHI17                 <= 0;
      reg__Cos_PHI17                 <= 0;
      reg__Sen_THETA32               <= 0;
      reg__Cos_THETA32               <= 0;
      reg__Sen_PHI32                 <= 0;
      reg__Cos_PHI32                 <= 0;
      reg__Sen_THETA                 <= 0;
      reg__Cos_THETA                 <= 0;
      reg__Sen_PHI                   <= 0;
      reg__Cos_PHI                   <= 0;
      reg__norm1                     <= 0;
      reg__norm2                     <= 0;
      reg__Sen_phi0                  <= 16'hE4F8;   // SETTAGGIO INIZIALE
      reg__Cos_phi0                  <= 16'h727C;   // SETTAGGIO INIZIALE
      reg__Sen_theta0                <= 16'hBECE;   // SETTAGGIO INIZIALE
      reg__u0                        <= 16'h5555;   // SETTAGGIO INIZIALE
      reg__v0                        <= 16'hAAAA;   // SETTAGGIO INIZIALE
      reg__w0                        <= 16'hAAAA;   // SETTAGGIO INIZIALE
      uno_meno_w0quadro              <= 16'h8E39;   // SETTAGGIO INIZIALE
      u                              <= 16'h5555;   // SETTAGGIO INIZIALE
      v                              <= 16'hAAAA;   // SETTAGGIO INIZIALE
      w                              <= 16'hAAAA;   // SETTAGGIO INIZIALE
      done                           <= 0;
      loaded                         <= 0;
    end
   else
    if (~loaded)
      begin
        if (start)
          begin
            reg__norm1         <= norm1;
            reg__norm2         <= norm2;
            reg__Sen_THETA17   <= Sen_THETA17;
            reg__Cos_THETA17   <= Cos_THETA17;
            reg__Sen_PHI17     <= Sen_PHI17;
            reg__Cos_PHI17     <= Cos_PHI17;
            reg__Sen_phi0      <= Sen_phi0;
            reg__Cos_phi0      <= Cos_phi0;
            reg__Sen_theta0    <= Sen_theta0;
            reg__u0            <= u;
            reg__v0            <= v;
            reg__w0            <= w;
            done               <= 0;
            cntr                = 0;
            loaded             <= 1;
          end
      end
    else if (loaded)
      begin
        case (cntr)
```

```
0 : begin
 reg__Sen_THETA32   <=     reg__Sen_THETA17 * reg__norm1;
 reg__Cos_THETA32   <=     reg__Cos_THETA17 * reg__norm1;
 cntr                                        =  1;
end

1 : begin
 reg__Sen_PHI32     <=     reg__Sen_PHI17 * reg__norm2;
 reg__Cos_PHI32     <=     reg__Cos_PHI17 * reg__norm2;
 reg__Sen_THETA     <=     reg__Sen_THETA32 >> 16;
 reg__Cos_THETA     <=     reg__Cos_THETA32 >> 16;
 cntr                =     2;
end

2 : begin
 reg__Sen_PHI               <=     reg__Sen_PHI32 >> 16;
 reg__Cos_PHI               <=     reg__Cos_PHI32 >> 16;
 reg__w0_per_Cos_phi0       <=     reg__w0 * reg__Cos_phi0;
 reg__w0_per_Sen_phi0       <=     reg__w0 * reg__Sen_phi0;
 cntr                        =     3;
end



3 : begin
 reg__Sen_THETA_per_Cos_PHI        <=     reg__Sen_THETA * reg__Cos_PHI;
 reg__Sen_THETA_per_Sen_PHI        <=     reg__Sen_THETA * reg__Sen_PHI;
 cntr                               =     4;
end

4 : begin
 reg__temp1 <= reg__Sen_THETA_per_Cos_PHI[2*dim_sincos - 3:dim_sincos - 1] *
               reg__w0_per_Cos_phi0[2*dim_sincos - 3:dim_sincos - 1];
 reg__temp2 <= reg__Sen_THETA_per_Sen_PHI[2*dim_sincos - 3:dim_sincos - 1] *
               reg__Sen_phi0;
 cntr       = 5;
end

5 : begin
 reg__temp3 <= reg__Sen_THETA_per_Cos_PHI[2*dim_sincos - 3:dim_sincos - 1] *
               reg__w0_per_Sen_phi0[2*dim_sincos - 3:dim_sincos - 1];
 reg__temp4 <= reg__Sen_THETA_per_Sen_PHI[2*dim_sincos - 3:dim_sincos - 1] *
               reg__Cos_phi0;
 reg__temp5 <= reg__temp1[2*dim_sincos - 3:dim_sincos - 1] –
               reg__temp2[2*dim_sincos - 3:dim_sincos - 1];
 cntr       = 6;
end

6 : begin
 reg__temp1  <=     reg__u0 * reg__Cos_THETA;
 reg__temp2  <=     reg__v0 * reg__Cos_THETA;
 reg__temp6  <=     reg__temp3[2*dim_sincos - 3:dim_sincos - 1] +
                    reg__temp4[2*dim_sincos - 3:dim_sincos - 1];
 cntr        =     7;
end

7 : begin
 reg__temp3  <= reg__w0 * reg__Cos_THETA;
 reg__temp4  <= reg__Sen_THETA_per_Cos_PHI[2*dim_sincos - 3:dim_sincos – 1]
                * reg__Sen_theta0;
 u           <= reg__temp1[2*dim_sincos - 3:dim_sincos - 1] + reg__temp5;
 cntr         = 8;
end

8 : begin
 v           <= reg__temp2[2*dim_sincos - 3:dim_sincos - 1] + reg__temp6;
 w           <= reg__temp3[2*dim_sincos - 3:dim_sincos - 1] –
                reg__temp4[2*dim_sincos - 3:dim_sincos - 1];
 cntr         =  9;
end
```

```
         9 : begin
          w0quadro    <=     (w * w);
          cntr         =     10;
         end

         10: begin
          uno_meno_w0quadro  <= 16'hffff - w0quadro[2*dim_sincos - 3:dim_sincos - 1];
          done               <= 1;
          loaded             <= 0;
         end

       endcase
      end
 end

endmodule

                        // MODULO CHE EFFETTUA LA TRASLAZIONE //

module trasl(clk,reset,start,d,cosdir_u,cosdir_v,cosdir_w,
posout_x,posout_y,posout_z,done);

`include "const.v"

parameter dim_state = 4;

input  clk, reset, start;
input  [dim_d - 1:0] d;
input  [dim_cosdir - 1: 0] cosdir_u, cosdir_v, cosdir_w;
output [dim_pos - 1: 0] posout_x, posout_y, posout_z;
output done;


reg    [dim_state - 1:0] current_state,next_state;
reg    [dim_pos - 1: 0] posin_x, posin_y, posin_z;
reg    [dim_pos - 1: 0] posout_x, posout_y, posout_z;
reg    loaded;
reg    done;
reg    [dim_pos - 1: 0] d_per_cosdir_u, d_per_cosdir_v, d_per_cosdir_w;
reg    reset_M;
reg    start_M;
reg    done_M1, done_M2, done_M3;
reg    [dim_d - 1:0] d_in;
reg    [dim_pos - 1: 0] cosdir_u_in, cosdir_v_in, cosdir_w_in;
reg    [dim_pos - 1: 0] d_per_cosdir_u_out, d_per_cosdir_v_out, d_per_cosdir_w_out;
reg    pause;
wire   done_all_M;

assign  done_all_M = done_M1 & done_M2 & done_M3;

Mult16  M1 (clk, reset_M, start_M, done_M1, d_in, cosdir_u_in, d_per_cosdir_u_out);
Mult16  M2 (clk, reset_M, start_M, done_M2, d_in, cosdir_v_in, d_per_cosdir_v_out);
Mult16  M3 (clk, reset_M, start_M, done_M3, d_in, cosdir_w_in, d_per_cosdir_w_out);

always @ (posedge clk)
 begin
  next_state = 0;
   case (current_state)

     0 :
     begin                                  // RESETTA TUTTO !
       posout_x             <= 0;
       posout_y             <= 0;
       posout_z             <= 0;
       posin_x              <= 0;
       posin_y              <= 0;
       posin_z              <= 0;
       done                 <= 1;
       loaded               <= 0;
       reset_M              <= 1;
       start_M              <= 0;
       d_in                 <= 0;
       cosdir_u_in          <= 0;
       cosdir_v_in          <= 0;
```

```
 cosdir_w_in           <= 0;
 pause                 <= 1;
 next_state             = 1;
end

1 :
begin
 if (reset == 1)
  next_state = 0;
 else
  begin
   reset_M <= 0;
    if (loaded == 0)
     begin
      done <= 1;
       if (start)
        begin
          done <= 0;
          next_state = 2;
        end
       else
        next_state = 1;
     end
    else
     begin
      start_M <= 0;
      next_state = 4;
     end
  end
end

2:
begin           // CARICA I DATI IN INGRESSO NEI REGISTRI INTERNI DI TRASL
 reset_M        <=     0;
 d_in           <=     d;
 cosdir_u_in    <=     cosdir_u;
 cosdir_v_in    <=     cosdir_v;
 cosdir_w_in    <=     cosdir_w;
 next_state      = 3;
end

3:
begin           // CARICA I DATI IN INGRESSO NEI REGISTRI INTERNI DI MULT16
 start_M        <= 1;
 reset_M        <= 0;
 posin_x        <= posout_x;          // AGGIORNA LA POSIZIONE  X DEL FOTONE
 posin_y        <= posout_y;          // AGGIORNA LA POSIZIONE  Y DEL FOTONE
 posin_z        <= posout_z;          // AGGIORNA LA POSIZIONE  Z DEL FOTONE
 loaded         <= 1;
 next_state      = 1;
end

4:
begin           // ATTENDE CHE TUTTI E TRE I MOLTIPLICATORI ABBIANO FINITO
 if (done_all_M)
  next_state = 5;
 else
  next_state = 4;
end

5:
begin
 if (pause)
  next_state = 6;
 else
  next_state = 7;
end

6:
begin    // CARICA NEI REGISTRI DI TRASL I RISULTATI DELLA MOLTIPLICAZIONE
 d_per_cosdir_u <=     d_per_cosdir_u_out;
 d_per_cosdir_v <=     d_per_cosdir_v_out;
 d_per_cosdir_w <=     d_per_cosdir_w_out;
 pause          <=     0;
```

```
       next_state     =  5;
      end

      7:
      begin           // FA LA SOMMA TRA LA VECCHIA POSIZIONE E LA DISTANZA PERCORSA
       posout_x   <= posin_x + d_per_cosdir_u;
       posout_y   <= posin_y + d_per_cosdir_v;
       posout_z   <= posin_z + d_per_cosdir_w;
       pause      <= 1;
       loaded     <= 0;
       done       <= 1;
       reset_M    <= 1;
       next_state = 1;
      end

    endcase
  end

always @ (negedge clk)
 current_state <= next_state;
endmodule

                // MOLTIPLICATORE A 16 BIT DA 16 CICLI DI CLOCK //

module Mult16 (clk, resetb, start, done, ain, bin, yout16);

parameter     N = 16;
input  clk;
input  resetb;
input  start;
input  [N-1:0]ain;
input  [N-1:0]bin;
output [N-1:0]yout16;
output done;

reg    [2*N-1:0]      a;
reg    [N-1:0]b;
reg    [2*N-1:0]      yout;
reg    [N-1:0]yout16;
reg    done;

always @(posedge clk or posedge resetb)
 begin
  if (resetb)
   begin
   a <= 0;
   b <= 0;
   yout <= 0;
   yout16 <= 0;
   done <= 0;
   end
  else
   begin
    if (start)
     begin
      a    <= ain;
      b    <= bin;
      yout <= 0;
      done <= 0;
     end
    else
     begin
      if (~done)
       begin
        if (b != 0)
         begin
          if (b[0])
           begin
            yout <= yout + a;
           end
          b <= b >> 1;
          a <= a << 1;
         end
```

```
            else
              begin
                done <= 1;
                yout16 <= yout[2*N-1:N];
              end
           end
        end
    end
 end
endmodule

                  // GENERATORE PSEUDO – CASUALE DELLA VARIABILE E //

module rnd_e (seme, clk,reset,start,e,done);

`include "const.v"

input  clk, reset, start;
input  [15:0] seme;
output [dim_phi - 1:0] e;
output done;

reg    [dim_theta - 1:0] e;
reg    done;
reg    loaded;

reg loadseed;
reg sig;
reg [15:0] out;
reg RNDgo;

wire clock, reset;
wire [15:0] seme;

assign clock = clk;

mainPRNG rnd3 (reset, seme, RNDgo, loadseed, sig, out);

always @ (negedge clk or posedge reset)
 begin
   if (reset)
     begin
       sig       <= 0;       // il mpx carica il seme se sig = 1!
       e         <= 0;
       done      <= 0;
       loaded    <= 0;
       loadseed  <= 1;
       RNDgo     <= clock;
     end
   else
     begin
     sig <= 0;       // il mpx riceve in ingresso l'uscita di xor4
       if (~loaded)
         begin
           if (start)
             begin
               e           <= out;
               done        <= 0;
               loaded      <= 1;
               RNDgo       <= 0;
             end
         end
       else if (loaded)
         begin
           done    <= 1;
           loaded  <= 0;
           RNDgo   <= 1;
         end
     end
 end
endmodule
```

```
     // MODULO CHE CALCOLA L'ENERGIA DEL FOTONE E DETERMINA LA FINE DEL PROCESSO //

module stop (clk,reset,start,e_in,alt,done);

`include "const.v"

input  clk, reset, start;
input  [dim_e - 1:0] e_in;
output alt, done;

reg    alt, done;
reg    [2:0] count;
reg    [dim_e - 1:0] e;
reg    loaded;

always @ (negedge clk or posedge reset)
  begin
    if (reset)
      begin
        count   = 1;
        alt    <= 0;
        done   <= 0;
        e      <= 0;
        loaded <= 0;
      end
    else
      begin
        if (~loaded)
          begin
            if (start)
              begin
                e      <= e_in;
                done   <= 0;
                loaded <= 1;
              end
          end
        else
          begin
            if (count == 0)
              alt <= 1;
            else
              count   = count + 1;
              done   <= 1;
              loaded <= 0;
          end
      end
  end
endmodule

                    // FILE DELLE COSTANTI USATE DAL BLOCCO SINCOS //

parameter max_cos = 65535;

task sin_const;

begin
  angle[0]  = 14'h2000; prod[0]   = 16'hb504;
  angle[1]  = 14'h12e4; prod[1]   = 16'ha1e8;
  angle[2]  = 14'h9fb;  prod[2]   = 16'h9d13;
  angle[3]  = 14'h511;  prod[3]   = 16'h9bdc;
  angle[4]  = 14'h28b;  prod[4]   = 16'h9b8e;
  angle[5]  = 14'h145;  prod[5]   = 16'h9b7b;
  angle[6]  = 14'ha2;   prod[6]   = 16'h9b76;
  angle[7]  = 14'h51;   prod[7]   = 16'h9b75;
  angle[8]  = 14'h28;   prod[8]   = 16'h9b75;
  angle[9]  = 14'h14;   prod[9]   = 16'h9b74;
  angle[10] = 14'ha;    prod[10]  = 16'h9b74;
  angle[11] = 14'h5;    prod[11]  = 16'h9b74;
  angle[12] = 14'h2;    prod[12]  = 16'h9b74;
  angle[13] = 14'h1;    prod[13]  = 16'h9b74;
end

endtask
```

```
                    // FILE DELLE DIMENSIONI DEI VARI BUS //

// ************** Bus dimensions ************************************

parameter dim_all      = 16; // Word dimension initialy equals for all
parameter dim_pos      = dim_all;
parameter dim_d        = dim_all;
parameter dim_theta    = dim_all;
parameter dim_phi      = dim_all;
parameter dim_alpha    = dim_all;
parameter dim_e        = dim_all;
parameter dim_cosdir   = dim_all;
parameter dim_sincos   = dim_all + 1;

// ************* Data constants ***************************************

parameter max_e = 30000;
```

Bibliografia

1) Alex F. Bielajew – **Fundamentals of the Monte Carlo method for neutral and charged particle transport** – Department of Nuclear Engineering and Radiological Sciences, University of Michigan (2000).

2) Robley D. Evans – **The atomic nucleus** – McGraw-Hill Book Company (1955).

3) Glenn F. Knoll – **Radiation detection and measurement** – John Wiley & sons (1979).

4) Stephen Gasiorowicz – **Quantum physics** – John Wiley & sons (1974).

5) Kenneth S. Krane – **Introductory nuclear Physics** – John Wiley & sons (1987).

6) P. Horowitz, W. Hill – **The art of electronics** – Cambridge University Press (1996).

7) M. Masetti, I. D'Antone – **Elettronica digitale –** Zanichelli (1995).

8) Attix, Roesch, Tochlin – **Radiation Dosimetry** – Academic Press Inc. (1969).

9) Hammersley, Handscomb – **Monte Carlo Methods** – Chapman and Hall (1983).

10) R. M. M. Oberman – **Digital Circuits for Binary Arithmetic** – The Macmillian Press LTD (1979).

11) Jack E. Volder – Transaction on electronic computers – EC-8:330-334 (1959).